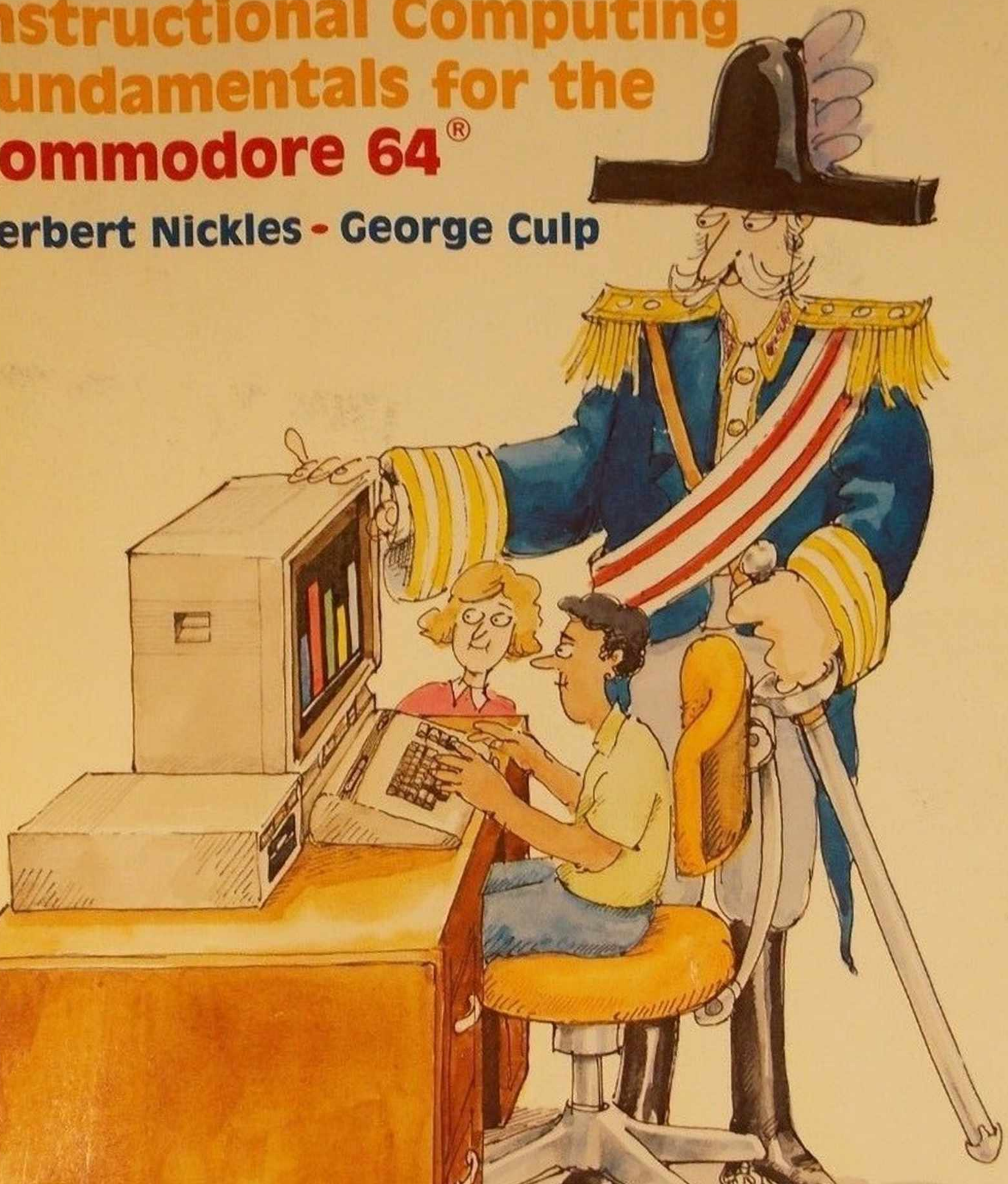
Instructional Computing Fundamentals for the Commodore 64®

Herbert Nickles - George Culp



Instructional Computing Fundamentals for the Commodore 64®

Instructional Computing Fundamentals for the Commodore 64®

Herbert L. Nickles
California State University, San Bernardino

George H. Culp
University of Texas at Austin

Brooks/Cole Publishing Company
Monterey, California

Brooks/Cole Publishing Company
A Division of Wadsworth, Inc.

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Nickles, Herbert L.
Instructional computing fundamentals for the Commodore 64.

(Brooks/Cole series in computer education)
Includes index.
1. Commodore 64 (Computer) — Programming. 2. Basic (Computer program language) I. Culp, George.
II. Title. III. Series.
QA 76.8.C64N53 1985 001.64'2 84-29222
ISBN 0-534-04662-2

Sponsoring Editor: Neil Oatley
Editorial Assistant: Gabrielle Bert
Production Service: Greg Hubit Bookworks
Manuscript Editor: Nick Murray
Interior Design: Katherine Minerva, Jamie Sue Brooks
Cover Design: Jamie Sue Brooks
Cover Illustration: Andrew Myer
Typesetting: Graphic Typesetting Service, Los Angeles
Printing and Binding: Malloy Lithographing, Inc.

For Daniel, Seth, Buck, Karyn,
and Tracy . . .
who make all things in life worthwhile.

Preface

The underlying premise of this book is our belief that teachers are by far the best potential source for the development of instructional computing materials. Thus, we have attempted to provide a *practical* book that presents the fundamentals of the BASIC programming language for the Commodore microcomputer, and explains how to apply them to the design and development of instructional computing programs. In fifteen years of teaching instructional computing courses to more than 2000 students, it has been our experience that, given these fundamentals, teachers are able to expand upon them and develop efficient programs to meet their specific needs.

Although the text may be used by teachers who have little or no computing experience, we want to emphasize that we are not attempting to teach general computer literacy: There is little mention of the history, architecture, or function of computers in society. Nor is this book for training computer programmers: Several language statements common to programming texts are omitted because their application is not typical of the *instructional* use of computers. Rather, in keeping with our goal of providing fundamentals, we present what a teacher needs to know in order to begin designing and developing instructional computing materials.

The book consists of eleven chapters and seven appendices, divided into two parts. In Part One, the first four chapters discuss the BASIC programming language statements and commands common to five areas of instructional computing use: problem solving, drill and practice, tutorial dialog, simulation, and testing. Chapter Five summarizes and reviews these statements and their applications. Chapters Six and Seven give relatively short example and model programs in each of the five areas. Chapter Eight presents a detailed examination of string functions and their applications to instructional computing. Chapter Nine discusses and demonstrates some simple uses of color and graphics as instructional techniques.

In Part Two, Chapters Ten and Eleven discuss the specific steps needed first to design and then to develop instructional computing materials. Ten criteria for the evaluation of instructional software are also included. The appendices include instructions for the general operation of the microcomputer and its disk system; a summary of commands and statements and their uses; answers to selected questions and problems given in the chapters; listings of programs given as solutions to certain problems; and instructions for making music with the Commodore 64.

As a matter of personal preference, some readers may wish to study Chapters Ten and Eleven on design and development before reading the chapters on BASIC. However, we believe that practical design and development can occur only after the working guidelines for the language are established. Thus, BASIC fundamentals are presented before we discuss design and development. We have found that presentation, demonstration, and discussion of the entire book requires 30–40 "class contact" hours. An equal or greater amount of student–terminal time outside of class is needed.

We discuss more than 40 programs, ranging from simple introductions to more sophisticated simulations. These programs are contained on a disk that is provided without cost by the publisher to adopters who order 25 or more copies of the book. We believe that the programs on the disk are very important and integral to the text; therefore, permission is hereby given to copy the disk for class-related use.

We would like to extend our appreciation to Carey Van Loon and Jami Hall of California State University, San Bernardino, for their excellent photographic assistance; to Dee Dee Watkins and her husband, Morgan, of the University of Texas at Austin; to Glenn Bull of the University of Virginia and Arthur Wiebe of Fresno Pacific College, who reviewed the entire manuscript; and to Carla Cramer, who keyed in over 30 of the programs. A special note of thanks is also due the people at Brooks/Cole Publishing Company who were involved with the development of this book.

Herbert L. Nickles
George H. Culp

Contents

Programs

Programs Discussed in the Text

Instructional Computing Fundamentals for the Commodore 64®

Introduction

This book describes an approach to using a common programming language, BASIC, for the design and development of instructional computing programs for Commodore 64 microcomputers. Its eleven chapters discuss certain fundamentals of BASIC and the design and developmental processes that provide a foundation for the production of instructional computing programs.

More than 100 books are available that teach BASIC (the *B*eginner's *A*ll-purpose *S*ymbolic *I*nstruction *C*ode, developed by John G. Kemeny and Thomas E. Kurtz at Dartmouth College). Although most of these books are very thorough in describing the language, they usually emphasize problem-solving applications. Our emphasis, on the other hand, is on instruction in the use of BASIC to design and develop materials for *instructional computing.*

Simply put, any use of computing techniques within the classroom may be broadly defined as instructional computing (sometimes known as *computer-assisted instruction* or *CAI*). Specifically, it includes:

1. *Problem solving,* in which computer programs are written to solve discipline-oriented problems.
2. *Drill and practice* on fundamental concepts using computer programs in a given discipline.

3. *Tutorial dialog,* in which computer programs provide "tutorlike" assistance in pointing out certain types of mistakes, providing review if needed, skipping areas in which proficiency is shown, and so on.
4. *Simulation,* in which computer programs allow manipulation and interpretation of certain elements related to given physical or social phenomena without the constraints of time, space, equipment, and environmental or logistical limits.
5. *Testing,* in which computer programs ask the questions, check the answers, and record the performance.

For our purposes, the term *instructional computing* includes all of these applications.

The use of BASIC. An introduction to some of the fundamentals of BASIC is provided in this book. This introduction is not intended to produce highly accomplished and skilled programmers. Rather, it gives only the fundamentals needed to write simple programs for instructional computing applications. Model programs are described that illustrate this use.

Although many different programming languages may be used in instructional computing, there are several reasons for using BASIC:

1. It is easy to learn and easy to use.
2. It is a common interactive language (see Section 1.3), available on large computer systems costing millions, medium-sized systems costing hundreds of thousands, minisystems costing tens of thousands, and small systems (commonly called *micros* or *personal* computers) costing a few hundred to a few thousand dollars.
3. It may be used in all of the applications of instructional computing described above.
4. It is one of the introductory computer languages used in most secondary and many elementary schools.
5. It is the most common language of microcomputers—an area of computer technology that is making the major impact on education in this decade.

Design. Following the introduction to BASIC, a method for designing instructional materials called the *systems approach* is outlined. This approach, in essence, is a logical, step-by-step process for identifying the tasks and activities needed in the production of validated instructional materials.

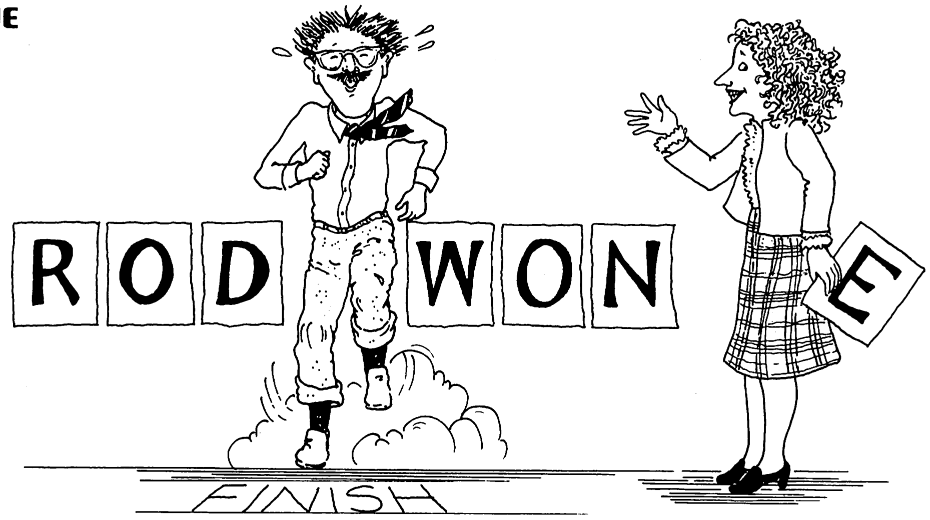
Development. The development of instructional computing programs by you is the ultimate goal of this book. Initially, the development phase overlaps the design phase, in which paper, pencil, and brain power are the principal ingredients. This process involves outlining the rationale, objectives, and instructional sequence of one or more instructional computing programs. After this program design is outlined on paper, it is translated into the BASIC programming code. The last step is to spend considerable time at a computer entering, testing, and refining what has been designed and developed on paper.

As a final introductory note, we emphasize that this book assumes no previous experience whatsoever with computers. On the other hand, the book does not provide detailed information on computers in general or how they operate. Rather, it introduces the ways and means by which the Commodore 64 microcomputer may be used within the instructional process.

Now, let us begin by getting down to the BASICs. . .

An Introduction to the BASIC Programming Language

Part One



A BASIC PROGRAM of My VERY OWN

"Nothing in life is to be feared. It is only to be understood."
Marie Curie

"In certain trying circumstances, urgent circumstances, desperate circumstances, profanity furnishes a relief denied even to prayer."
Mark Twain

"If at first you don't succeed, you are running about average."
M. H. Alderson

Think About This (for Fun)
Rearrange the letters of NEW DOOR to form one word. [Note: Answers to Think About This (for Fun) questions may be found in Appendix C.]

Think About This (Seriously)
Does a computer possess intelligence?

1.1 Objectives

For the successful completion of this chapter, you should be able to:

1. List five general applications of instructional computing (Introduction).
2. Define two ways in which computers may be accessed (Section 1.3).
3. List the steps necessary to "boot up" (power up) a computer system (Appendix A).
4. State how a BASIC program may be entered on that system after the "booting up" (Section 1.6 and Appendix A).
5. Define what (not who) composes a BASIC program (Section 1.4.1).
6. Distinguish between BASIC statements and commands (Sections 1.4.1–1.4.2).
7. Define the action of the following BASIC commands: NEW, RUN, LIST, and SAVE (Section 1.4.2).
8. Define and give at least one example of both a *numeric* variable and a *string* variable (Section 1.4.3).
9. Describe the use of commas and semicolons in BASIC for purposes other than punctuation (Section 1.4.4).
10. Define the purpose and give at least one example of the following BASIC statements: PRINT, INPUT, LET, and END (Sections 1.5.1–1.5.4).
11. Describe three simple techniques for editing BASIC programs (Section 1.6 and Appendix B, Section B.5).

1.2 Computer Use: A Brief History and Rationale

Electronic computers have been in use since the late 1940s. In the period from 1948 to 1965, they were used primarily for what their name implies: computing, or "number crunching" as it is sometimes called. Starting about the mid-sixties, however, educators began experimenting with applications of computers in the instructional process that involved more than just computing.

In the decade following, this use expanded, and, just as computers have become ingrained in our society, instructional computing is becoming commonplace in our schools. (These points may be emphasized by the fact that since 1975 over 1,500,000 microcomputers have been purchased, many for home or school use.)

It is very important to recognize that computers are not replacing teachers! The fundamental principle underlying the use of computers—regardless of the profession using them—is that they are incredibly fast and accurate tools, and they allow people to do certain activities in a manner that has never before been possible. Thus, the use of computers in instruction is basically (no pun intended) that of *supplemental* applications. Computers allow teachers and students to do certain educational processes faster, with greater accuracy, and in a manner not possible before they came on the scene.

Computer programs can be very helpful in providing patient, routine drill on fundamental concepts, in generating and grading tests in a given discipline, and in many other applications. In any of these cases, the most effective programs are those designed by teachers: the professionals in the field who are aware of what is to be taught and how to teach it. As yet, there is no computer program that can lead an intelligent and sensitive discussion on any given abstract concept. There are no teachers out of a job because they have been replaced by a computer! That is something worth remembering.

1.3 Access to Computers

A computer is an extremely fast and accurate processor of data. In the simplest sense, most common computer systems may be viewed as four units connected electronically:

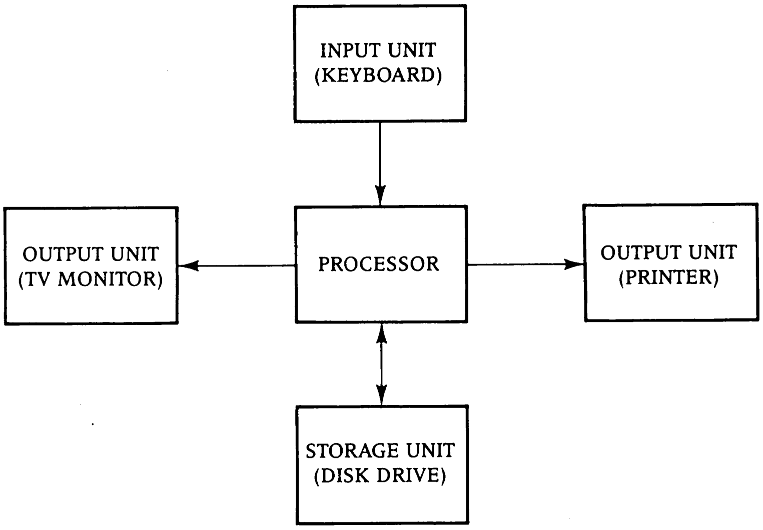
1. An *input* unit (such as a computer terminal keyboard), through which data is entered.
2. A *processor* unit, which stores the data input and processes it electronically.
3. An *output* unit (such as a computer terminal screen or printer), which shows the results of processing the data input.
4. A *data storage/retrieval* unit (such as a disk drive), which stores data on and retrieves data from some magnetic medium (such as a floppy disk).

Figure 1.1 shows these units in block form.

Until the late 1960s, the primary means of access involved punching program statements, data, and commands onto computer cards. This "batch" of cards was read (input) by a card reader and eventually a printout (output) of the program "run" was retrieved. This type of access is commonly referred to as *batch access*, or *batch processing*.

Since the early 1970s, there has been a very strong trend toward accessing computers via computer terminals. In the simplest sense,

FIGURE 1.1
Components of a computer system



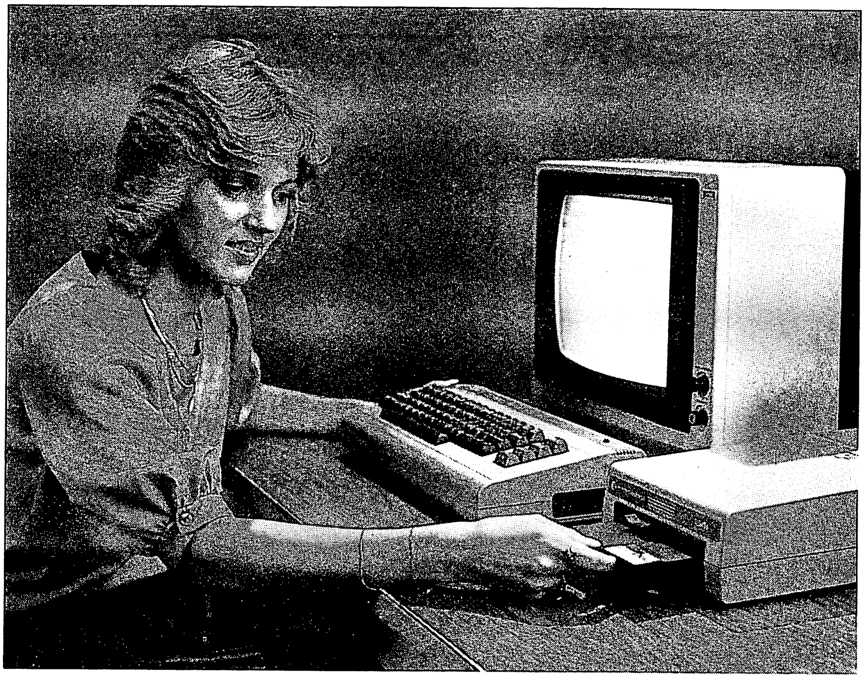
a terminal consists of a keyboard, similar to that of a typewriter, for input of statements, data, commands, and so forth, with output displayed either on a cathode ray tube (CRT) screen or paper *(hardcopy)* at the terminal. This type of access is known as *interactive* (a user is interacting directly with the computer or a program), or *timesharing* (there may be literally scores of terminals in remote locations "sharing the time" of one computer). In most instances, the terminal is connected to the computer via standard telephone lines.

Microcomputers are an exception to this. Here the computer, terminal, display, and other components are usually provided as a unit small enough to fit on a desk top (Figure 1.2). There are no telephone connections or sharing of computer time. These features make the unit more portable, less prone to equipment failure, less expensive, and, consequently, well suited to the classroom.

For our use here, only microcomputers are discussed. The examples and assignments in the book assume that the reader has access to a Commodore 64 microcomputer with BASIC, one floppy disk drive, a video monitor or TV, and at least 64K of random access memory (RAM).

It is very important that the reader, particularly the reader new to microcomputers, become familiar with the processes needed to access (use) the system. This first involves gaining confidence in "booting up" the system. Refer to Appendix A for a step-by-step procedure to accomplish this.

FIGURE 1.2
Commodore 64 microcomputer system: keyboard, monitor, disk drive, and floppy disk.



1.4 A Bit About BASIC Before Beginning

There are a few general points about BASIC that should be made early. Consider these as some of the "rules of the game" to follow for BASIC.

1.4.1 Statements

A BASIC program may be defined as at least one "instruction" performed (executed) by the computer. Instructions are written in the form of *statements*. These statements are words (often verbs), such as PRINT, INPUT, and so on, that make some degree of sense to both a user and the computer. (Of course, the computer has been programmed by people to "understand" these words.) The length of a BASIC statement must not exceed 80 total characters, including spaces.

BASIC statements are always numbered, generally by tens (10, 20, 30, etc.). They could be numbered 1, 2, 3, and so on, but no additional statements could be inserted into the program, say, between statements numbered 1 and 2. Statements can be inserted between lines numbered 10 and 20 (11, 12, etc.). Thus, the numbering convention is usually in increments of ten.

1.4.2 Commands

BASIC commands issue specific information to the computer system *about* the program. For example, the command NEW instructs the system to prepare for a new BASIC program to be entered at the terminal by "erasing" any program statements that are currently in the system's memory. The command LIST will produce a listing of the BASIC statements comprising the program in memory.

The command RUN executes (RUNs) the BASIC statements in their increasing numerical sequence unless one of those statements transfers the execution to another part of the program. (This process is called *branching* and will be discussed later.) The command SAVE "filename",8 instructs the system to save on the disk the program in memory under the name "filename". The program is stored on a floppy disk placed in the disk drive. (The "filename" may be just about any name, but short, descriptive names should be considered.)

1.4.3 Variables

Nearly all BASIC programs described in this text will include values that may *vary* as the program is executed (RUN). These values, called *variables*, could be students' names, test scores, responses for correct or incorrect answers, and so forth.

A variable, in BASIC, may be represented (named) simply by any letter of the alphabet. However, the name could be as long as eighty characters *if* the *first* character in the name is a letter, and the rest of the name contains only letters or numbers. However, only the first *two* characters are considered by the Commodore. Realistically, variable names should be short and descriptive. Initially (for simplicity), we will use only one or two characters for variable names in our example programs. Later, where appropriate, longer variable names that better describe their values may be used. For our purposes here, there are two types of variables:

1. *Numeric.* The value of the variable is always numeric: 1.0, 2, 110.5, -3.1365, and so on.
2. *String (or alphanumeric).* The value of the variable may be alphabetic characters or numbers or a mixture of both. Such values are *always* enclosed in quotation marks: "ABCDEF", "CS395T", "JOHN JONES", "NOW IS THE TIME", and so on.

A dollar sign ($) is added to the name of the string variable to distinguish it from a numeric variable. N$, A1$, Z9$, and FI$ all represent string variable names, while N, A1, Z9, and FI all represent numeric variable names.

Examples:

```
A = 123
```

(The *numeric* variable named A has a value of 123 assigned to it.)

```
A$ = "ABC"
```

(The *string* variable named A$ has a value of ABC assigned to it.)

1.4.4 Commas (,) and Semicolons (;)

Commas and semicolons have specific uses in BASIC. They can be used in the normal fashion as punctuation marks, or they can be used to instruct the system to display information in special ways. For example, every so often in a BASIC program there may be a need to have information printed in columns. Suppose a list of student names, test score averages, and final numeric grades were to be displayed (PRINTed). Assume the values are stored in the variables N$, T, and F, respectively. The BASIC statement

```
PRINT N$,T,F
```

would display this information in columns ten spaces apart from the start of the first value to the start of the second value, and so on. Here, the comma acts as an automatic tabulator of ten spaces. Thus, any line can have "fields" of display starting at column 1, column 11, and so on. This can be useful when certain types of information, for example, name and grade for a series of students, is to be displayed. (See, for example, Sections 1.5.1 and 1.5.3.)

If one wished the above information to be *closepacked* (printed without any separating spaces), the semicolon would be used in place of the comma. In essence, then, the comma, when *not* used as a punctuation mark, instructs the system to tab ten spaces before printing; similarly, the semicolon instructs the system not to skip *any* spaces before printing.

These and other examples of their use will be shown shortly, but for now be aware that the comma and semicolon can have special meanings when not used as punctuation.

1.5 BASIC Statements for This Chapter

1.5.1 Statement—PRINT

Purpose

Displays (PRINTs) information at the computer terminal. This information may be text, numeric variable values (see Section 1.5.3), or string (alphanumeric) variable values (see Section 1.5.3). When text is to be displayed, it must be enclosed in quotation marks (") in the PRINT statement. (*Note:* BASIC statements, such as PRINT, will be shown in this book in upper case for clarity. However, statements may also be typed in lower case, if so desired.)

Example:

```
PRINT "Hello, What's your first name"
```

Result of Execution:

```
Hello, What's your first name
```

In addition, certain shorthand conventions are available for the Commodore 64 microcomputer. For example, the shorthand for the statement PRINT is the question mark (?). Thus, PRINT statements may be literally typed as:

```
10 ? "Hello, What is your first name"
```

The system will automatically interpret the ? as the word PRINT.

Example:

```
10 A=123
20 A$="ABC"
30 ? A,A$
```

Result of Execution:

```
123          ABC
```
←10 spaces→

1.5.2 Statement—INPUT

Purpose

Allows numeric or alphanumeric information to be entered (INPUT) into a BASIC program during its execution. The information is entered through the terminal keyboard and is assigned to a variable specified by the program author. The variable will have the assigned value until changed by another INPUT or LET (see below) statement for that variable.

Examples:

`INPUT N` (for numeric information)
`INPUT N$` (for alphanumeric information)
`INPUT "Your name is";N$` (text with statement)
`INPUT "Your age is";N` (text with statement)

Note: Most BASIC systems automatically display a question mark when the INPUT statement is executed. In computer terms, the question mark (?) is called the *input indicator,* or *prompt.* Program execution is stopped until the RETURN key is depressed. Also, note that the use of quotes discussed earlier for string variables (Section 1.4.3) is not required when string information is INPUT. The dollar sign ($) instructs the system that *any* input will be assigned as a string variable. Also note that text may be enclosed in quotation marks as part of the INPUT statement. However, a semicolon *must* follow the last quotation mark before the variable name.

Program Example:

```
10 PRINT "Hello, What's your first name"
20 INPUT N$
30 PRINT N$ " is a nice name."
40 END
```

Result of Execution:

```
Hello, What's your first name
```
?SAMMY (the question mark appears as a prompt, SAMMY is typed, and the RETURN key depressed)
```
SAMMY is a nice name.
```

1.5.3 Statement—LET

Purpose

Assigns values to variables. This action may be "direct," as in LET X = 20 (X would have a value of 20), or it may be "indirect," as in LET X = (2*Y)/3 (X would have a value equal to the result of dividing 3 into the product of 2 times the value of Y). The "*" is the symbol (character) used for multiplication; the "/" is the symbol used for division. It may also be used to assign alphanumeric values, as in:

```
LET A$ = "Here's the answer!"
```

Note: In most BASIC systems, the term LET is optional, so the statement X = 20 would be equivalent to LET X = 20. Also, note here that assignment to a string variable requires the use of quotes. The string content *must* be enclosed in quotes.

Program Example:

```
10 LET N$ = "JIM JONES"
20 LET T1 = 100
30 FI = 89 (Note: LET is omitted)
40 PRINT "STUDENT","TEST 1","FINAL"
50 PRINT "-------","------","-----"
60 PRINT N$,T1,FI
70 END
```

Result of Execution:

```
STUDENT        TEST 1         FINAL
-------        ------         -----
JIM JONES      100            89
```

What would happen if the commas in statements 40–60 were replaced by semicolons? (*Note:* Answers to this and other questions found within the book are supplied under their respective chapter and section numbers in Appendix C.)

1.5.4 Statement—END

Purpose

Ends program execution. On many systems, the END statement is not required for program execution. (See also the statement STOP in Appendix B.)

1.5.5 PROGRAM 1: Years-to-Days Conversion

Note: In this book we will be discussing more than 30 sample programs. Most of the programs are named in the sequence of their discussion: PROGRAM 1 comes before PROGRAM 2, and so on.

The statements discussed thus far can be combined to make a program. But what is the program to do? Some stage of program design must be defined that illustrates the use of these statements. Arbitrarily, then, the program is designed to:

1. Ask for a person's (the program user's) name (PRINT).
2. Store the name entered in a string variable (INPUT).
3. Greet that person with the name entered (PRINT).
4. Skip a line so that the screen is not too crowded (PRINT).
5. Ask for the person's age in years (PRINT).
6. Store the age entered in a numeric variable (INPUT).
7. Convert the age in years to the age in days (LET).
8. Skip a line to avoid crowding on the screen (PRINT).
9. Display this age in days (PRINT).
10. End the program (END).

Note: In creating any new program, the following sequence should be followed:

1. Enter NEW
2. Enter the program statements
3. Enter SAVE "<filename>",8
4. RUN the program to test it
5. Make any needed modifications
6. When satisfied, enter SAVE "@0:<same filename>",8

Step 6 above *replaces* the old version of the program <filename> with the new (modified) version. If a copy of *both* versions is desired, then simply enter SAVE "<different filename>",8. Also, the statements comprising a program in memory may be seen by entering LIST. If a specific line (statement) number or a range of line numbers is desired, enter LIST <line number> or LIST <beginning line–ending line>, respectively.

RUN from disk and refer to the listing and run of PROGRAM 1.

PROGRAM 1

```
10 "HELLO, WHAT'S YOUR FIRST NAME"
```

Statement 10 displays a greeting and asks for the user's first name.

```
20 INPUT N$
```

Statement 20 automatically displays a "?" and waits for input from the terminal keyboard. Whatever is typed is assigned as a value to the variable N$ when the RETURN key is depressed.

```
30 PRINT "HOWDY, " N$
```

This value is displayed with the text, "Howdy, " in statement 30. Carefully note that the variable N$ is not *enclosed within the quotation marks. (Why is the blank space included within the quotation marks?)*

```
40 PRINT
```

Statement 40 prints a blank line, and statement 50 requests the user's age in years.

```
50 PRINT "TELL ME, , ,WHAT IS YOUR AGE IN YEARS";
60 INPUT A
```

Statement 60 automatically displays a "?" and waits until some number is typed and the RETURN key depressed. This value is assigned to variable A.

```
70 D=A * 365
```

Statement 70 assigns a value to variable D equal to the value of A times 365 (converting years to days).

```
80 PRINT
```

Statement 80 prints a blank line.

```
90 PRINT "WELL, " N$ ", YOU HAVE BEEN BREATHING
100 PRINT "FOR AT LEAST " D " DAYS!"
```

The value of variable D, with appropriate text, is then displayed in statements 90 and 100. Again, carefully note that only *the text to be displayed,* not *the variable names (N$ and D), is enclosed within the quotation marks.*

```
110 PRINT , "BYE-BYE, " N$
120 END
```

Statement 110 skips over ten spaces and displays a farewell. Statement 120 ends program execution.

1.6 Editing BASIC Programs

Most BASIC systems have some means by which programs may be edited. For example, a PRINT statement with a misspelled word or typographical error may be corrected by editing. Three simple editing techniques are:

1. *Cursor keys (e.g., →).* Two cursor keys (CRSR) are located on the lower right area of the keyboard. With the additional use of the shift key, pressing one of these keys allows movement of the cursor in any of four directions. Thus, the cursor may be located wherever an error is found or insertion is desired. Pressing the INST/DEL key (upper right keyboard area) will DELete the character to the left of the cursor. If the SHIFT key is pressed with this key, INSerTions may be performed. Practice using these keys on some sample statements. They make editing a relatively simple task once you have mastered their use.

2. *Retyping the statement.* A statement may be replaced by simply retyping the line number followed by the correct statement.
3. *Deleting lines.* A statement may be deleted entirely by typing the line number *only* and then depressing the RETURN key.

Although these are only three simple techniques for editing, they will get you started and can be extremely useful. Knowledge of the CRSR keys on the keyboard and their use will be particularly helpful.

1.7 Posers and Problems

(*Note:* Many of the "Posers and Problems" given in this book may be entered and run as programs. Where possible, this should be done, since it will be of help in arriving at the solutions. As a last resort, or to check your work, refer to Appendix C. Problems headed by an asterisk (*) may be considered as more difficult problems.)

1. Correct any errors found in the following BASIC statements:

```
10 PRIMT "Hello
20 PRIMT What's your height in inches"
30 INPUT
40 M = 2.54 *
50 PRINT You are M centimeters tall!
60 FINISH
```

2. Mentally determine the value of X in each of the following if Y = 6.
   X = 25
   X = (2*Y)/3
   X = Y
   X = (2*Y)/(3*Y)
   X = (Y*Y)/(Y*2)
3. Why is a space included within the quotation marks before or after the variable names in statements 90, 100, and 110 in PROGRAM 1? Are the variables N$ and D actually enclosed in quotation marks as they might appear in statements 90 and 100?
4. Note the different position of the two question marks in the sample RUN of PROGRAM 1. What caused the difference? (*Hint:* Carefully examine statements 10 and 50.)
5. Modify PROGRAM 1 to output the user's age in "heartbeats" (use H as the variable), assuming a pulse rate of 72 beats per minute (and 60 minutes per hour, 24 hours per day).

6. What would result if the following statements were executed?

```
10 A$ = "NAME"
20 B$ = "SCORE"
30 C$ = "AVERAGE"
40 PRINT A$,B$,C$
50 END
```

7. What would result if the following statements were executed? (Assume you input your own name and weight.)

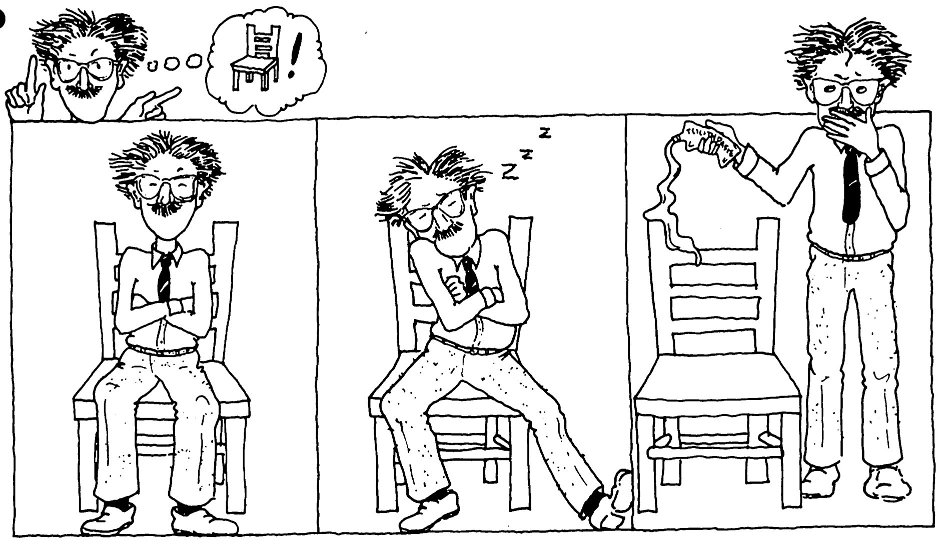
```
10 PRINT "Your first name is";
20 INPUT N$
30 PRINT "Your weight in pounds is";
40 INPUT P
50 K = P/2.2
60 Z = P * 16
70 PRINT,"Wow, " N$ "!"
80 PRINT "That's only " K " kilograms, but, gee,"
90 PRINT,"it's " Z " ounces!"
100 END
```

*8. Write a program that converts a temperature in Celsius to a temperature in Fahrenheit. The user should enter the temperature for conversion from the keyboard. (*Hint:* The formula for conversion is F = (C * 9/5) + 32.)

*9. Write a program that converts two variables, cups and ounces, into ounces. For example, 2 cups and 3 ounces equal 19 ounces.

*10. Write a program that inputs two string variables, first name and last name, and prints out a salutation of your choice using the person's full name.

Chapter Two



Now Tell It Where To Go and What To Do with It

"Even if you're on the right track, you'll get run over if you just sit there."
Will Rogers

"Man's mind stretched to a new idea never goes back to its original dimensions."
Oliver Wendell Holmes

Think About This (for Fun)
What do you sit on, sleep on, and brush your teeth with?

Think About This (Seriously)
Can computer programs teach?

2.1 Objectives

For the successful completion of this chapter, you should be able to:

1. Define the purpose and give at least one example of each of the BASIC statements REM, GOTO, IF-THEN, and ON-GOTO (Sections 2.2.1–2.2.4).
2. Define the purpose and give at least one example of each of the BASIC functions RND(0) and INT (Section 2.3).
3. Define the purpose of the BASIC command LOAD (Section 2.4).
4. Define the purpose and give at least one example of the PRINT TAB statement (Section 2.5.1).
5. Alone and unafraid, boot up a microcomputer system (Appendix A).
6. Design, enter, and RUN a BASIC program that includes the statements discussed in Chapters 1 and 2.

2.2 BASIC Statements for This Chapter

2.2.1 Statement—REM

Purpose

Used as a REMinder or REMark to document the listing of BASIC programs. That is, REM gives a means by which internal notes may be made in the program listing. These notes will provide information about the program, such as which variables are used and their purpose (commonly called a *dictionary of variables*), and will identify special program routines or strategies, separating the program into segments so that the program listing is easy to read. The REM statement is not executed during a program RUN; thus, the only time this is displayed is after a LIST command.

Example:

```
REM ===The Variable A is the age in years===
```

(The only purpose of characters such as = = = is to make the REM statements stand out in the listing of the program.)

2.2.2 Statement—GOTO

Purpose

Unconditionally transfers (branches) program execution to the specified statement number.

Example:

```
GOTO 100
```

Note: Use of the GOTO statement in programs should be minimized. If this statement is used excessively, following the "flow" of a program in a "frame-by-frame" fashion can become very difficult.

2.2.3 Statement—IF-THEN

Purpose

Conditionally transfers (branches) program execution to the specified statement number if, and only if, the defined variable relationship is *true.*

Examples:

```
IF X = 1 THEN 100
```

(Transfer to statement 100 will occur only if X is *equal* to 1.)

```
IF Y <> Z THEN 100
```

(Transfer to statement 100 will occur only if the value of Y is *not equal* to the value of Z.)

```
IF A <= 2 THEN 100
```

(Transfer to statement 100 will occur only if the value of A is *less than* or *equal* to 2.)

```
IF A >= 2 THEN 100
```

(Transfer to statement 100 will occur only if the value of A is *greater than* or *equal* to 2.)

```
IF A$ = "YES" THEN 100
```

(Transfer to statement 100 will occur only if the value of A$ is *equal* to (the same value as) the character string YES.)

Also, note that IF-OR-THEN and IF-AND-THEN statements are possible:

```
IF A < 1 OR A > 10 THEN 100
```

(Transfer to statement 100 will occur only if the value of A is *less than* 1 or *greater than* 10.)

```
IF A = 2 AND B = 3 THEN 100
```

(Transfer to statement 100 will occur only if the value of A is *equal* to 2 *and* the value of B is *equal* to 3.)

Combinations of string and numeric variables also may be used:

```
IF Z$ = "YES" AND C > 8 THEN 100
```

(Transfer to statement 100 will occur only if the value of Z$ is *equal* to the string YES *and* the value of C is *greater than* 8.)

Make special note that many statements such as PRINT, (LET), and others to be seen in later chapters can be included in the IF-THEN statement. If the condition defined by the IF-THEN statement is *true,* the statement included will be executed; otherwise, execution continues with the next numbered statement.

```
IF G > 69 THEN PRINT "You passed!"
```

(If the value of S is greater than 69, then "You passed!" will be PRINTed.)

```
IF Z$ = "N" THEN A = 0
```

(If the value of Z$ is equal to the string N, then the numeric variable A is set to zero.)

2.2.4 Statement—ON-GOTO

Purpose

Transfers (branches) program execution to a specified statement number based on the rounded value of a variable or numerical relationship.

Example:

```
ON X GOTO 100,300,600
```

(Transfer to statement 100 will occur if the value of X is 1; transfer to statement 300 will occur if this value is 2; transfer to statement 600 will occur if this value is 3. If X is 0 or greater than 3 in the example above, execution continues with the first statement following the ON-GOTO.)

This example of the ON-GOTO is equivalent to the following three IF-THEN statements:

```
IF X = 1 THEN 100
IF X = 2 THEN 300
IF X = 3 THEN 600
```

By using the ON-GOTO statement, the same instructions can be given to the system by just one statement:

```
ON X GOTO 100,300,600
```

2.3 Some Very BASIC Functions

Functions in BASIC are essentially mathematical routines that either come with the computer system (as a library of routines or functions) or may be defined by the user. Once a function has been defined, it may be used over and over again without the bother of writing out the entire routine.

Two of the most common library functions that are used in instructional computing applications are RND(0) and INT. When executed, the RND(0) function automatically gives some random numeric value between 0.0 and 0.999999999. The INT function reduces any number with a decimal fraction (called a *real* number) to a whole number (called an *integer*).

By using a combination of these functions in BASIC statements, it is possible to generate random numbers within any range desired. This may be used to generate different values for questions containing numbers, randomly selecting questions by number from a "bank" of questions, randomly branching to specified line numbers using ON-GOTO statements, and so on. The following illustration shows how this combination may be used to generate numbers in the range of 1–10, inclusive.

Suppose a BASIC statement looked like this:

```
X = INT(10 * RND(0) + 1)
```

and suppose RND(0) comes up with a random value of 0.58. BASIC is set up so that numerical operations enclosed in parentheses are performed first. Thus, the steps the system follows in computing the value of X would be:

1. 10 * 0.58 = 5.8
2. 5.8 + 1 = 6.8
3. INT of 6.8 = 6

Thus, X will have a value of 6 in this example. What would be the value of X if RND(0) = 0.999999999? What would be the value of X if RND(0) = 0.01? What is the *range* of random numbers that could result from the statement:

```
X = INT((100 * RND(0) + 1) * 10) / 10
```

What statement would generate random numbers in the range of 1.00 to 100.00, inclusive? (*Hint:* Note the two decimal places.) How is an integer value changed to a real value containing two decimal places? Answer: By dividing the integer by 100.0. What statement would produce random numbers in the range of 5–95, inclusive?

A general formula may be derived that will give any desired range of positive random numbers:

```
N = INT((H - L + 1) * RND(0) + L)
```

where N is the random number generated and H and L are the highest and lowest numbers, respectively, in the desired range. For example, a range of random numbers is desired between 100 and 25.

Highest number = 100
Lowest number = 25
100 − 25 + 1 = 76

The statement to generate this range of random numbers is:

```
N = INT(76 * RND(0) + 25)
```

2.4 Modification of Existing Programs

In Chapter 1, Problem 5 asked the reader to modify PROGRAM 1 to output (PRINT) the number of heartbeats equivalent to a user's age in years, assuming there were 72 beats per minute. To do this, it is necessary to:

1. Retrieve PROGRAM 1 from the disk
2. Make the modifications
3. Save the modified version of PROGRAM 1 as PROGRAM 2.

[By saving the *modified* program as PROGRAM 2 (SAVE "PROGRAM 2", 8) both the old version (PROGRAM 1) and the new version (PROGRAM 2) are on the disk. If only the new version is wanted, the same name (PROGRAM 1, in this case) should be used (SAVE "@0:PROGRAM 1",8).]

2.4.1 PROGRAM 2: Adding Heartbeats

Recall that PROGRAM 1 was created by first typing NEW to erase any program in memory and then entering each line, statement by statement. The program was SAVEd; then RUN was entered to test it. Once a program has been SAVEd, it may be retrieved for use or modifications by the command

```
LOAD "<name>",8
```

(where <name> is the name of the program)

If any changes are made that are to be permanent in the program, the command SAVE "@0:<name>",8 must be used.

In summary, we have the following commands:

Command	Example	Action
NEW	NEW	Clears memory of statements
RUN	RUN	Executes statements in memory
LOAD "<name>",8	LOAD "PROGRAM 1",8	LOADs the program "<name>" from the disk to memory
LIST	LIST	LISTs the entire program
LIST nn	LIST 10	LISTs line nn
LIST nn–mm	LIST 10–100	LISTs lines nn–mm, inclusive
SAVE "<name>",8	SAVE "PROGRAM 1",8	SAVEs a NEW program in memory on the disk as <name>
SAVE "@0:<name>",8	SAVE "@0:PROGRAM 1"	*Replaces* old version of PROGRAM 1 with new version

Note: A "shorthand" method to automatically LOAD and RUN a program from disk is:

`LOAD "<name>",8:` (simultaneously press SHIFT and RUN/STOP keys)

RUN from disk and refer to listing and run of PROGRAM 2.

PROGRAM 2

PROGRAM 1 is first LOADed and then LISTed. Statements 101–109 are entered.

```
10 PRINT "HELLO, WHAT'S YOUR FIRST NAME"
20 INPUT N$
30 PRINT "HOWDY, " N$
40 PRINT
50 PRINT "TELL ME, , ,WHAT IS YOUR AGE IN YEARS";
60 INPUT A
70 D = A * 365.25
```

Note that statement 70 is reentered to reflect a more accurate value for the number of days per year (365.25 versus 365).

```
80 PRINT
90 PRINT "WELL, " N$ ", YOU HAVE BEEN BREATHING"
100 PRINT "FOR AT LEAST " D " DAYS!"
101 REM ============
102 REM MODIFICATIONS ADDED BELOW
103 REM ============
104 PRINT
105 H = D * 24 * 60 * 72
```

Statement 105 converts the age in days, D, to heartbeats, H, since there are 24 hours per day, 60 minutes per hour, and 72 heartbeats per minute.

```
107 PRINT "AND THAT'S A LIFETIME OF"
108 PRINT H "TOTAL HEARTBEATS!"
```

Statements 107 and 108 display the value of the variable H, along with appropriate text.

```
109 PRINT
110 PRINT ,"BYE-BYE, " N$
120 END
```

The program is then RUN.

```
RUN
HELLO, WHAT'S YOUR FIRST NAME
?HERBIE

HOWDY, HERBIE

TELL ME, , ,WHAT IS YOUR AGE IN YEARS?46

WELL, HERBIE, YOU HAVE BEEN BREATHING
```

```
FOR AT LEAST 16801.5 DAYS!

AND THAT'S A LIFETIME OF
1.74197952E+09 TOTAL HEARTBEATS!
          BYE-BYE, HERBIE
```

The program is SAVEd as PROGRAM 2.

```
SAVE "PROGRAM 2",8
```

Note: In the RUN of the program, the value of H is expressed as 1.74197952E+09. This is the method in which the system displays a value of 1,741,979,520. It is also the system's way of expressing scientific notation; that is, 1.74197952 × 10^9. This amounts to one billion, seven hundred forty one million, nine hundred seventy nine thousand, five hundred and twenty heartbeats! Though easily broken, 'tis still a powerful muscle!

2.5 Incorporating the New Statements

The content design of any BASIC program is at the discretion of its author (programmer). The program can be as simple or as complex as the author desires. For example, BASIC may be used in trivial Fahrenheit to Celsius temperature conversions or in sophisticated modeling of population dynamics. The point is that a program does only what an author has designed it to do—nothing more or less. However, for any program, regardless of its simplicity or complexity, the author must first outline the design and "flow" of the program. On that note, the following program (PROGRAM 3) is designed only to illustrate a use of the statements discussed in this unit.

2.5.1 PROGRAM 3: Appropriate Responses

The program will ask a question and give only one chance for a correct answer. "Appropriate" responses will be made for either a correct or incorrect answer. The program will then ask a final question related to age. The user will be informed if the answer is too low or too high. For answers that are too high, an additional comment will be randomly selected from three choices. The question will be repeated until the correct answer is given.

The previous statements outline what the program is designed to do. The BASIC statements needed to accomplish this are shown in the program listing. (Now, try to relax when you see the "long" listing of the program. Think about what each statement instructs the system to do and mentally follow its execution.)

RUN from disk and refer to the listing and run of PROGRAM 3.

PROGRAM 3

```
10 REM     PROGRAM 3
20 REM ===============
30 REM ASK SOME QUESTION
40 REM GET AN ANSWER AND
50 REM CHECK FOR ACCURACY
60 REM ===============
70 PRINT "WHAT STATE FOLLOWS ALASKA"
```

Statements 70 and 80 (the first executed statements) print out the question.

```
80 PRINT "IN TOTAL LAND AREA";
80 INPUT R$
```

Statement 90 displays a question mark (note where it is displayed), waits for input, and stores it in the string variable R$.

```
100 IF R$ = "TEXAS" THEN 130
110 PRINT ,"NOPE, IT'S TEXAS!"
120 GOTO 140
130 PRINT TAB( 3) "YEEE-HAAA! YOUR ANSWER I6 CORRECT!"
```

(Note: From now on, reference to statements will sometimes be made by number only. That is, statement 100, for example, will simply be referred to as 100.) 100 checks for the value of R$ to be equal to TEXAS. If it is, transfer to 130 occurs. If not, 110 is executed. Then 120 causes transfer to 140, skipping 130, the response for the correct answer.

```
140 PRINT
```

140 prints a blank line.

```
150 REM ================
160 REM ASK ANOTHER QUESTION
170 REM AND CHECK FOR HIGH OR
180 REM LOW ANSWER INPUT
180 REM ================
200 PRINT "WHAT WAS THE PERPETUAL AGE"
210 PRINT "OF THE LATE JACK BENNY";
```

200 and 210 print the next question.

```
220 INPUT R
```

220 displays a question mark, waits for input from the terminal, and stores it in variable R.

```
230 IF R < 39 THEN 290
```

230 checks for the value of R to be less than 39. If it is, transfer to 290 occurs. If not, execution continues to 240, where R is checked for a value greater than 39. If it is, transfer is to 320. If not, 260 and 270 are executed. (Statements 230–270 essentially say that if R is not greater than 39 nor less than 39, then it must be equal to 39.)

```
240 IF R > 39 THEN 320
250 REM ===THEN INPUT EQUALS 39===
260 PRINT
270 PRINT TAB(3)"DIDN'T LOOK IT..." TAB(25)"DID HE???"
280 GOTO 490
280 PRINT ,"TOO LOW..."
```

290 (from 230, if it were a true statement) tells the user that the answer was too low, and 310 returns to the question (140).

```
300 REM ===REPEAT THE QUESTION===
310 GOTO 140
320 PRINT ,"TOO HIGH,,,"
```

320 (from 240, if it were a true statement) tells the user that the answer was too high. Then 380 gives a random value for X between 3 and 1, inclusive.

```
330 REM ===============
340 REM GET A RANDOM COMMENT FOR
350 REM ANY ANSWER THAT IS HIGH
360 REM THEN REPEAT THE QUESTION
370 REM ===============
380 X = INT (3 * RND (0) + 1)
390 ON X GOTO 400,420,440
```

390 transfers execution to 400 if $X = 1$; 420 if $X = 2$; or 440 if $X = 3$. These statements print an additional "comment" and return the execution to 140 (by 410, 430, or 450).

```
400 PRINT "NOW THAT IS OLD!"
410 GOTO 140
420 PRINT "ARE YOU TRYING TO BE CRUEL?"
430 GOTO 140
440 PRINT "HAVE YOU NO SYMPATHY?"
450 GOTO 140
460 REM ===============
470 REM END THE PROGRAM
480 REM ===============
490 PRINT
500 PRINT
510 PRINT ,"BYE-BYE, FRIENDS, , ,"
520 END
```

If R is not less than or greater than 39 (statements 230–270), 280 transfers execution to 490, which prints a blank line, as does 500. A farewell is printed by 510 and the program ends at 520.

```
RUN
WHAT STATE FOLLOWS ALASKA
IN TOTAL LAND AREA?CALIFORNIA
          NOPE, IT'S TEXAS

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?33
          TOO LOW, , ,

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?43
          TOO HIGH, , ,
HAVE YOU NO SYMPATHY?
```

```
WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?42
          TOO HIGH. . .
NOW THAT IS OLD!

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?39

   DIDN'T LOOK IT. . .     DID HE???

        BYE-BYE FRIENDS. . .

RUN
WHAT STATE FOLLOWS ALASKA
IN TOTAL LAND AREA?TEXAS
   YEEE-HAAA! YOUR ANSWER IS CORRECT!

WHAT WAS THE PERPETUAL AGE
OF THE LATE JACK BENNY?39

   DIDN'T LOOK IT. . . DID HE???

          BYE-BYE FRIENDS. . .
```

Carefully note statements 130 and 270 in the program listing. Each of these incorporates the PRINT TAB statement. This statement allows an automatic tabulation of a defined number of spaces before display or PRINTing occurs.

2.6 Posers and Problems

1. What is the difference between the variables R and R$ in PROGRAM 3?
2. What would be the result if statement 390 in PROGRAM 3 read ON X GOTO 440,400,420?
3. Modify statements 400, 420, and 440 in PROGRAM 3 to give comments of your choosing.
4. What should be done to PROGRAM 3 so that it would ask for your age instead of Jack Benny's?
5. What should be done to PROGRAM 3 in order to select a random comment from five choices instead of three?
6. What changes should be made to PROGRAM 3 in order to ask for the third largest state by land area instead of the second?
7. How should PROGRAM 3 be modified to ask for the user's first name at the start of the program and then refer to the user by name when "BYE-BYE . . . " is executed in statement 510?

8. Add some REM statement to PROGRAM 1 so that the variables are made clearer to someone looking at the listing of the program for the first time.
9. Write a statement that will randomly give a value for variable X that is between 200 and 50, inclusive.
10. What is the range of numbers that could randomly be generated by the statement:

```
X = INT(25 * RND(0) + 5)
```

*11. Write a program that asks for the user's height in inches and then prints "TALL" if the user is over six feet, "SHORT" if under five feet, or "AVERAGE" if between five and six feet, inclusive.
*12. Write a program that inputs the lengths of the sides of a triangle as variables A, B, and C (largest side last) and determines if it is a right triangle. (*Hint:* For right triangles, C ↑ 2 = A ↑ 2 + B ↑ 2. The "up arrow" (↑) is the system's way to "raise to the power of."
*13. Write a program that inputs a number and prints "THREE" if it is a 3, "SIX" if it is a 6, "NINE" if it is a 9, or "NEITHER 3, 6, nor 9" if it is not equal to either 3, 6, or 9.

Chapter Three

Take a Ride on the Loop-D-Loop

"Don't put off for tomorrow what you can do today, because if you enjoy it today you can do it again tomorrow."
James A. Michener

Think About This (for Fun)
What is the exact opposite of not in?

Think About This (Seriously)
Is the use of computers in instruction just another "educational fad"?

3.1 Objectives

For the successful completion of this chapter, you should be able to:

1. Define and give at least one example of each of these BASIC statements: PRINT "CLR/HOME key", DATA-READ, RESTORE, and FOR-NEXT (Sections 3.2.1–3.2.4).
2. Define the purpose of and give at least one example of "multiple statements per line" (Section 3.2.2).
3. Enter and RUN each of the BASIC programs used as statement examples in this chapter.
4. Design, enter and RUN a BASIC program that contains the statements discussed in Chapters 1–3.

3.2 BASIC Statements for This Chapter

3.2.1 Statement—PRINT "CLR/HOME Key"

Purpose

PRINT "CLR/HOME key" CLearS (erases) all display and places the cursor in the upper left corner of the monitor screen. This use is particularly appropriate in instructional computing since it allows information, examples, questions, and so on, to be displayed in a frame-by-frame fashion. *Note:* The CLR/HOME is a key on the upper right portion of the keyboard. It is used in a program as a statement by:

1. Typing the word PRINT
2. Holding down the SHIFT key
3. Pressing the double quote (")
4. Pressing the CLR/HOME key
5. Pressing the double quote (")
6. Releasing the SHIFT key
7. Pressing the RETURN key

Note: A shaded "heart" will appear enclosed in the quotation marks.

Example:

```
PRINT "♡"
```

Example:

```
10 PRINT "♡" : PRINT : PRINT "Hello!"
```

In this example, three "actions" are executed:

1. The screen is erased and the cursor positioned in the upper left corner (PRINT "CLR/HOME key").
2. One line is skipped (PRINT)
3. "Hello!" is displayed on the next line (PRINT "Hello!").

This example introduces another shorthand method in BASIC: the use of multiple statements per line. In the example shown, three separate BASIC statements are combined on *one* line. Note that the colon is used as a "delimeter," or separator, for each statement. Use of multiple statements per line makes more efficient use of the memory of the microcomputer system. However, it has the disadvantage of making the LISTing of the program more difficult to read in terms of its design and flow. Therefore, the program examples in this book will make minimum use of multiple statements on one line. However, remember that as you become more experienced, this is a handy shorthand method. (Also remember that the total length of a multiple-statement line must not exceed 80 characters.)

Enter and RUN the following example:

```
10 PRINT "♡" : PRINT : ?, "Enter your name"
20 PRINT TAB(15); : INPUT N$
30 PRINT, : ? "Hi, "N$"!"
```

[Could *all* of these statements (10–30) be combined on one line?]

3.2.2 Statement Pair—DATA-READ

Purpose

DATA allows information (numeric or string) to be stored in a program for use at various stages throughout its execution. The pieces of information are generally referred to as *data elements*, with each element separated (*delimited*) by a comma. READ assigns the defined value of a data element to a specified variable and "moves" a data "marker" or "pointer" to the next data element. The data type (numeric or string) *must* match the variable type (numeric or string). For example, "ABC" cannot be assigned to a numeric variable!

One additional important note should be made in regard to the elements in the DATA statements: *Never* place a comma at the end of the DATA statement. The system may take the space character following the comma as the next data element!

Example:

```
10 PRINT "♡"
20 DATA "Help! I'm ",45," and sinking fast!"
30 READ P1$
40 PRINT P1$
50 READ A
60 PRINT A
70 READ P2$
80 PRINT P2$
90 PRINT : INPUT "Please press the "RETURN" key";Z$
100 PRINT : PRINT
110 PRINT P1$;A;P2$
```

What caused the display to change from vertical to horizontal? Did the values of the variables change from one display to the next? After entering and RUNning this example, delete lines 50 and 70. Retype line 30 as

```
30 READ P1$,A,P2$
```

then RUN the program again. Why is there no difference in the display?

Note that this example also demonstrates one method to "hold" the screen display for an indefinite period *until* the RETURN key is pressed. Also note that spaces are included within the quotation marks of the string DATA elements to prevent close packing when the variable values are PRINTed.

Example:

```
10 PRINT "♡" : X = 0
20 DATA 10,15
30 READ N
40 X = X + N
50 PRINT "The value of N is "N TAB(30)"The value of X is "X
60 READ N
70 X = X + N
80 PRINT "The value of N is "N TAB(30)"The value of X is "X
```

Why did the values of variables N and X change? Note that variable X is, in effect, a cumulative total of the values assigned to variable N. (After mentally tracing the program execution, enter and RUN the above example to check your mental interpretations.)

3.2.3 Statement—RESTORE

Purpose

Moves the data pointer to the first data element in the first DATA statement.

Example:

```
10 PRINT "♡" : X = 0
20 DATA "I'm a ","prisoner of ","love!"
30 READ A$,B$,C$
40 PRINT,, A$;B$;C$
50 X = X + 1
60 PRINT "The value of X is "X
70 IF X = 2 THEN 100
90 GOTO 30
100 PRINT : PRINT "Sigh ..."
```

Enter and RUN the above program and note what happens. Add the statement 80 RESTORE and RUN again. Note the results.

What is the position of the data pointer after statement 30 has been executed but before the RESTORE statement is added? What caused the error message in the first RUN? What caused the program to stop execution after the statement 80 RESTORE was added? What would the program do if statement 50 were deleted (after 80 RESTORE was included in the program)? Think this through before RUNning; otherwise, remember that depressing the RUN/STOP and RESTORE keys simultaneously will halt the execution of a "runaway" program!

Note: On many BASIC systems, statements such as 10 X = 0 (as in the programs above, for example) are not needed because all variables are automatically "initialized" (set) to zero. However, it is good programming practice to initialize variables to zero in any program.

3.2.4 Statement Pair—FOR-NEXT

Purpose

Defines the number of times (loops) a series of consecutive BASIC statements are to be repeated. FOR defines the variable used as a counter for the repeats and the lower and upper limits of the count. NEXT increases the variable count by one (or the defined STEP size) and checks to see if the upper limit of the FOR is exceeded. If not, execution is transferred to the statement immediately following the FOR statement. If the upper limit is exceeded, execution is transferred to the statement immediately following the NEXT statement.

The variable names in the loop defined by a FOR-NEXT must be identical. Note that the start and/or limit of the loop may be defined by variable names, as in FOR X = Y TO Z.

Also note that loops may be defined in *decreasing* order

```
FOR I = 10 TO 1 STEP -1
⋮
NEXT I
```

In this example, the counter begins at 10 and decreases to 1 in increments (steps) of −1.

Examples:

```
10 INPUT "Enter your first name";N$
20 FOR C = 1 TO 20
30    PRINT "♡" : PRINT TAB(C);
40    PRINT N$
50 NEXT C
```

For some interesting effects, replace statements 20 and 30 above with combinations of the following:

```
20 FOR C = 1 TO 30 STEP 5
30    PRINT TAB(C);
```

Also, experiment with different STEP increments.

```
10 PRINT "Let's get rolling . . ."
20 FOR X = 2 TO 12 STEP 2
```

```
30    PRINT "Here's "X" and its cube:"
40    PRINT X,X*X*X
50 NEXT X
60 PRINT,"That's all . . ."

10 A = 10
20 B = -10
30 FOR C = A TO B STEP -2
40    PRINT C;" ";
50 NEXT C
60 PRINT "And we counted backwards by twos. . ."
```

(Mentally trace the execution, and then enter and RUN each program.)
Note: Indentation is for clarity.

3.3 Incorporating the New Statements

With the addition of the statements in this chapter, a BASIC program may be designed that provides more of a utility than the earlier programs. One of the many uses of computer programs involves searching a list of information (commonly called a *data base*) for key elements that may be specified by a user. For example, data bases may be searched for financial accounts that are overdue by 30, 60, or 90 days; address lists may be searched for ZIP codes; employee rolls may be searched for persons who have special deductions; and so on. The following program illustrates one search technique.

3.3.1 PROGRAM 4: Searching for a Range of Values

This program contains a list of DATA elements representing pairs of hypothetical names and scores. This list is to be searched for scores that fall within a specified maximum and minimum range. A list of names and scores that are within this range is printed. Following this, the user is given an option to do another search.

What will this require as the program is mentally designed? In outline form, there must be at least:

1. Prompts to get the maximum and minimum scores (PRINTs).
2. Entry of these scores (INPUTs).
3. A loop (FOR) to:
   a. READ the DATA.
   b. Check (IF-THEN) to see if the current score read is the last data element in the list and, if not, to see if it is in the maximum-minimum range.

c. Display (PRINT) the name and score if in the range.
d. Continue the search (NEXT).

4. A prompt for another search option (PRINT and INPUT).
5. Movement of the data pointer back to the first data element if another search is to be done (RESTORE), followed by repetition of the total process (GOTO).
6. A list of names and scores in the program (DATA).
7. An end to the program (PRINT and END).

RUN from disk and refer to the listing and run of PROGRAM 4.

PROGRAM 4

```
10 REM ===============
20 REM  PROGRAM 4 DESCRIPTION
30 REM ===============
40 REM  KEY SEARCH OF 50 OR LESS DATA ELEMENT PAIR.
50 REM  PROGRAM SEARCHES FOR A MINIMUM-MAXIMUM RANGE OF
SCORES.
60 REM  DATA ELEMENTS ARE IN SEQUENCE: NAME,SCORE.
70 REM  LAST SEQUENCE OF DATA ELEMENTS IS "X",0
80 REM ===============
90 REM VARIABLE DICTIONARY
100 REM ============
110 REM N$ - HYPOTHETICAL NAME
120 REM S  - HYPOTHETICAL SCORE
130 REM M1 - MAXIMUM SCORE
140 REM M2 - MINIMUM SCORE
150 REM I  - LOOP COUNTER
160 REM F  - COUNTER FOR THE NUMBER OF MATCHES FOUND
170 REM ===========
180 REM SET THE COUNTER TO ZERO, CLEAR
190 REM THE SCREEN, AND GET THE RANGE SOUGHT.
200 REM ===========
```

Statement 220 clears the screen and places the cursor at the upper left-hand corner of the screen.

```
210 F = 0
```

210 initializes variable F to zero.

```
220 PRINT "♡"
```

Statement 200 clears the screen and places the cursor at the upper left-hand corner of the screen.

```
230 PRINT "MAXIMUM SCORE";
240 INPUT M1
250 PRINT "MINIMUM SCORE";
260 INPUT M2
```

Statements 230–260 obtain the maximum and minimum range of scores desired by the user.

```
270 PRINT "NAMES WITH SCORES IN THE RANGE:" M2 "-" M1
280 PRINT "NAME","SCORE"
PRINT "----","-----"
```

Statements 270–290 PRINT a heading for the list.

```
300 REM ===============
310 REM DO THE SEARCH LOOP
320 REM ===============
330 FOR I = 1 TO 50
```

Statements 330–420 define a FOR-NEXT loop.

Statement 330 assigns variable I as a counter for the loop, sets its initial value to 1, and defines its upper limit as 50. (In other words, the loop will be performed a maximum of 50 times.)

```
340 READ N$,S
```

Statement 340 READs a DATA element pair from the data list beginning at statement 560 and assigns the values read to N$ and S, respectively. (The first time READ is executed, N$ would have a value of SUE, S would be equal to 67, and the data pointer would be moved to BOB in preparation for execution of the next READ statement.)

```
350 REM ===END OF DATA LIST?===
360 IF N$ = "X" THEN 480
```

Statement 360 checks the value of N$. If this value is equal to the character X, execution is transferred to 460. (X has been arbitrarily defined as the last name in the list of data. This gives the program a way of identifying when the last data element pair has been read.)

```
370 REM ===IS THE SCORE "S" OUTSIDE THE RANGE?===
380 IF S > M1 or S < M2 THEN 420
```

Statement 380 checks the value of S, the S just READ, to see if it is outside the maximum-minimum range (greater than M1 or less than M2). If so, execution is transferred to 420, where the loop is repeated if the value of I + 1 does not exceed the defined upper limit of 50.

```
390 PRINT N$,S
```

If the value of S falls within the range (S is neither greater than M1 nor less than M2), statement 390 PRINTs the current value of N$ and S.

```
400 REM ===COUNT THE MATCHES FOUND===
410 F = F + 1
```

Statement 410 increases the counter F by one for each match that is found.

```
420 NEXT I
```

Statement 420 repeats the loop if it has not yet occurred 50 times.

```
430 REM ===============
440 REM END OF CURRENT SEARCH
450 REM ===============
450 PRINT "THIS SEARCH FOUND " F " MATCH(ES)."
```

Statement 460 may be executed from either of two sources:

1. From statement 360, if the last data element has been READ. (Note: *N$ would have a*
value of X.)

2. From statement 420, if the NEXT I (the value of I + 1) exceeds the limit defined by the FOR in statement 330.

```
470 REM ===GIVE OPTION TO DO ANOTHER SEARCH===
480 PRINT "DO YOU WISH ANOTHER SEARCH (Y OR N)";
480 INPUT Z$
500 IF Z$ <>"Y" THEN 1000
```

Statements 480–500 give the option for another search. If the INPUT value for variable Z$ is not equal to Y (for Yes), transfer is to 1000, and the program ENDs.

```
510 RESTORE
```

Otherwise, statement 510 RESTOREs the data pointer to the first data element.

```
520 GOTO 210
```

Statement 520 then transfers execution back to 210 so that another search may take place.

```
530 REM ==============
540 REM     DATA LIST
550 REM ==============
560 DATA "SUE",67,"BOB",55,"JACK",98,"MARY",88,"STAN",50,"ROB",7
570 DATA "LETA",77,"ALEX",66,"SUSAN",85,"MARIA",99,"FRAN",70
580 DATA "BOBBIE",100,"CHARLES",64,"BILLY",56,"MAGGIE",86
580 DATA "DONNA",91,"YANCY",77,"TRACY",99,"KAREN",100,"BUCK",90
600 REM ===ROOM FOR MORE DATA===
999 DATA    "X",0
1000 PRINT "*** SEARCH COMPLETED ***"
1010 END

MAXIMUM SCORE?100
MINIMUM SCORE?90
NAMES WITH SCORES IN THE RANGE: 90-100
NAME       SCORE
----       -----
JACK       98
MARY       99
MARIA      99
BOBBIE     100
DONNA      91
KAREN      100
BUCK       90


THIS SEARCH FOUND 7 MATCH(ES).
DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
MAXIMUM SCORE?60
MINIMUM SCORE?50
```

```
NAMES WITH SCORES IN THE RANGE: 50-60
NAME       SCORE
----       -----
BOB        55
STAN       50

THIS SEARCH FOUND 2 MATCH(ES).
DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
MAXIMUM SCORE?77
MINIMUM SCORE?77
NAMES WITH SCORES IN THE RANGE: 77-77
NAME       SCORE
----       -----
LETA       77
YANCY      77

THIS SEARCH FOUND 2 MATCH(ES).
DO YOU WISH ANOTHER SEARCH (Y OR N)?Y
MAXIMUM SCORE?49
MINIMUM SCORE?0
NAMES WITH SCORES IN THE RANGE: 0-49
NAME       SCORE
----       -----
THIS SEARCH FOUND 0 MATCH(ES).
DO YOU WISH ANOTHER SEARCH (Y OR N)?N
*** SEARCH COMPLETED ***
```

3.3.2 Additional Comments on PROGRAM 4

DATA statements may be included anywhere in a BASIC program. They are not executed as, for example, a PRINT or INPUT statement would be. Their only use is to contain data (information) that is to be READ and assigned to variables. The DATA statements in PROGRAM 4 are placed near the end of the program so that additional DATA statements may be added if desired. The last data element pair, 'X' and 0, is given the highest number possible for a DATA statement (999 in this case). Thus, additional DATA statements could be inserted between statements 600–999 if there were a need to add more data.

In RUNning the program, how could the DATA in PROGRAM 4 be searched for only *one* user-specified score, listing all the names with that score?

3.4 A Time-Saving Technique

There may be times when a user wishes to SAVE both the "old" and "new" versions of a program. The "new" (modified) version of a program may be SAVEd by simply giving it a new (unique) name when the SAVE command is issued. We did this earlier in Chapter 2 when PROGRAM 1 was modified (with the heartbeats) and SAVEd as PROGRAM 2.

To illustrate this, PROGRAM 5 will be created and SAVEd. Then it will be modified and SAVEd as PROGRAM 5A. This means that both the old (PROGRAM 5) and the new (PROGRAM 5A) programs will be available for future use.

3.4.1 PROGRAM 5: Subtraction Drill

Arbitrarily, this new program will be a drill on subtraction practice. Then it will be modified to be a drill on addition practice.

Run from disk and refer to the listing and run of PROGRAM 5.

PROGRAM 5

```
10 REM PROGRAM 5 DESCRIPTION
20 REM ===============
30 REM THIS PROGRAM PROVIDES DRILL ON SUBTRACTION
40 REM USING RANDOM NUMBER GENERATORS. IT ALSO
50 REM INTRODUCES THE USE OF MULTIPLE STATEMENTS
60 REM ON ONE LINE, PAUSING, AND "FRAMING" TECHNIQUES.
70 REM BY SIMPLE MODIFICATION, THE PROGRAM MAY BE
80 REM CONVERTED TO ADDITION, MULTIPLICATION OR DIVISION.
90 REM ===VARIABLE DICTIONARY===
100 REM A - THE DIFFERENCE BETWEEN 2 RANDOM NUMBERS
110 REM C - A COUNTER FOR THE NUMBER OF CORRECT ANSWERS
120 REM I - A COUNTER USED TO "HOLD" THE SCREEN DISPLAY
130 REM N1 - A RANDOM NUMBER, 1-12, INCLUSIVE
140 REM N2 - ANOTHER RANDOM NUMBER, 1-12, INCLUSIVE
150 REM P - THE NUMBER OF PROBLEMS SELECTED BY THE USER (3-15)
160 REM Q - A COUNTER FOR THE QUESTION LOOP
170 REM R - THE USER'S RESPONSE TO A QUESTION
180 REM Z$ - A "DUMMY" INPUT TO HOLD SCREEN DISPLAY
    INDEFINITELY
190 REM ==============
200 REM CLEAR SCREEN, CENTER, DISPLAY A TITLE
210 REM FOR ABOUT 3 SECONDS; THEN CLEAR SCREEN AGAIN
220 REM ==============
```

Statements 10–220 give a brief description of the program and variables used.

```
230 PRINT "♡" : FOR I = 1 TO 11 : PRINT : NEXT I
240 PRINT TAB(5)"S U B T R A C T I O N  D R I L L"
```

Statements 230–240 clear the screen and show the title to be displayed.

```
250 REM NOTE MULTIPLE STATEMENTS PER LINE IN 230 AND 260
260 FOR I = 1 TO 3000 : NEXT I
```

Statement 260 uses multiple statements per line (FOR-NEXT) to "count" to 3000. In effect, this gives a pause of about three seconds before the screen is erased by statement 320.

```
270 REM ===============
280 REM LET THE USER SELECT THE NUMBER OF PROBLEMS
290 REM WITHIN ARBITRARY LIMITS OF AT LEAST 3
300 REM BUT NO MORE THAN 15, THIS DEFINES THE QUESTION LOOP.
310 REM ===============
320 PRINT "♡" : FOR I = 1 TO 11 : PRINT : NEXT I
330 PRINT "HOW MANY PROBLEMS DO YOU WANT (3-15)";
340 INPUT P
```

Statements 330–340 allow a user to INPUT the number of problems to practice. Arbitrarily, the limit has been defined as at least 3, but no more than 15 problems.

```
350 REM ===CHECK TO BE WITHIN RANGE===
360 IF P < 3 OR P > 15 THEN 320
```

Statement 360 checks to see if the value of P is within the defined range of problems possible.

```
370 REM ===============
380 REM BEGIN THE QUESTION LOOP
390 REM ===============
400 FOR Q = 1 TO P
```

Statements 400–710 define the "question loop."

```
410 PRINT "♡" : FOR I = 1 TO 11 : PRINT : NEXT I
420 REM ===GENERATE THE TWO RANDOM NUMBERS===
430 N1 = INT(12 * RND(0) + 1)
440 N2 = INT(12 * RND(0) + 1)
450 REM ===============
460 REM ARBITRARILY, WE WANT N1 TO BE GREATER THAN
470 REM N2, SO WE'LL CHECK IT.
480 REM ===============
490 IF N1 < N2 THEN 430
```

Statements 420–490 generate two random numbers in the range of 1 to 12 and ensure that the first number is equal to or larger than the second number.

```
500 REM ===SET "A" EQUAL TO THE ANSWER===
510 A = N1 - N2
```

Statement 510 defines the correct answer to be equal to the difference between the two random numbers and assigns this value to variable A.

```
520 REM ===PRINT OUT THE PROBLEM===
530 PRINT TAB(12) N1 "-" N2 "=";
540 INPUT R
```

Statements 530–540 display the problem and get INPUT for the answer.

```
550 IF A = R THEN 690
```

Statement 550 checks to see if the answer INPUT (variable R) is equal to the correct answer (variable A). If so, transfer is to statement 690.

```
560 PRINT : PRINT TAB(7)"NO, THE CORRECT ANSWER IS";A
570 REM ===============
580 REM LET THE USER DETERMINE WHEN TO CONTINUE
590 REM IN THAT THE QUESTION WAS MISSED AND MIGHT
```

```
600 REM NEED TO BE "STUDIED."
610 REM ===============
620 PRINT : INPUT " PRESS THE RETURN KEY TO CONTINUE";Z$
630 GOTO 710
```

If the correct answer is not given, statements 560–630 display it. This is "held" on the screen until the RETURN key is pressed. Transfer is then to statement 710, which continues the question loop.

```
640 REM ===============
650 REM FOR EVERY PROBLEM ANSWERED CORRECTLY:
660 REM CLEAR SCREEN, CENTER REINFORCER, HOLD ABOUT 1 SECOND,
670 REM INCREASE CORRECT COUNTER BY 1, AND CONTINUE LOOP.
680 REM ===============
690 PRINT "♡" :FOR I = 1 TO 11:PRINT:NEXT I:PRINT TAB(17)"O.K.!"
700 FOR I = 1 TO 1000 : NEXT I : C = C + 1
```

Statements 690–700 clear the screen, give a positive comment at the approximate center of the screen, hold it there for about one second, increase a "number correct" counter by one, and continue the question loop.

```
710 NEXT Q
720 REM ==============
730 REM THIS IS THE END OF THE QUESTION LOOP, SO LET'S
740 REM TELL HOW MANY CORRECT ANSWERS WERE GIVEN.
750 REM ==============
760 PRINT "♡" : FOR I = 1 TO 11 : PRINT : NEXT I
770 PRINT "YOU ANSWERED" C "QUESTION(S) CORRECTLY!"
780 END
```

Statements 760–780 clear the screen and show the user the number of problems correctly answered before ending the program.

```
RUN
         SUBTRACTION DRILL

HOW MANY PROBLEMS DO YOU WANT (3-15)100
HOW MANY PROBLEMS DO YOU WANT (3-15)3

         5 - 2 = ?3
         O.K.

         25 - 2 = ?24

NO, THE CORRECT ANSWER IS 23

PRESS THE RETURN KEY TO CONTINUE

         25 - 9 = ?16
                 O.K.

YOU ANSWERED 2 QUESTION(S) CORRECTLY!
```

3.4.2 PROGRAM 5A: Addition Drill (A Modification of PROGRAM 5)

Careful examination of the listing of PROGRAM 5 shows that the minimum changes needed are statements 240, 510, and 530. After PROGRAM 5 has been LOADed from the disk, the following is entered:

```
240 PRINT "A D D I T I O N  D R I L L"
510 A = N1 + N2
530 PRINT TAB(12) N1 " + " N2 " = ";
SAVE "PROGRAM 5A",8
```

Of course, there are other changes that could and should be made. The REM statements need to reflect that the program is addition drill. Statements 450–500 are not necessary since the first number could be smaller than the second, and perhaps statements 430–440 might be altered to give a larger range of numbers. However, the statements and SAVE command shown above are all that is needed to change the program from subtraction to addition drill. Thus, a new program, PROGRAM 5A, based upon another program, PROGRAM 5, has been created and SAVEd without the time and trouble required to completely retype the new version.

RUN from disk and refer to the listing of PROGRAM 5A. (*Note:* A sample RUN is not included.)

PROGRAM 5A

```
10 REM PROGRAM 5A DESCRIPTION
20 REM ===============
30 REM
40 REM THIS LISTING OF PROGRAM 5A SHOWS MORE THAN THE
50 REM MINIMUM CHANGES MENTIONED IN THE TEXT. OTHER
60 REM CHANGES INCLUDE DIFFERENT RANDOM NUMBERS
70 REM (STATEMENTS 430-440) AND DELETION OF
80 REM STATEMENTS 450-490.
90 REM ===VARIABLE DICTIONARY===
100 REM A  - THE DIFFERENCE BETWEEN 2 RANDOM NUMBERS
110 REM C  - A COUNTER FOR THE NUMBER OF CORRECT ANSWERS
120 REM I  - A COUNTER USED TO "HOLD" THE SCREEN DISPLAY
130 REM N1 - A RANDOM NUMBER, 1-12, INCLUSIVE
140 REM N2 - ANOTHER RANDOM NUMBER, 1-12, INCLUSIVE
150 REM P  - THE NUMBER OF PROBLEMS SELECTED BY THE USER (3-15)
160 REM Q  - A COUNTER FOR THE QUESTION LOOP
170 REM R  - THE USER'S RESPONSE TO A QUESTION
180 REM Z$ - A "DUMMY" INPUT TO HOLD SCREEN DISPLAY
    INDEFINITELY
190 REM ==============
200 REM CLEAR SCREEN, CENTER, DISPLAY A TITLE
210 REM FOR ABOUT 3 SECONDS; THEN CLEAR SCREEN AGAIN
```

```
220 REM ==============
230 PRINT "♡" : FOR I = 1 TO 11 : PRINT : NEXT I
240 PRINT TAB(8)"A D D I T I O N  D R I L L"
250 REM NOTE MULTIPLE STATEMENTS PER LINE IN 230 AND 260
260 FOR I = 1 TO 3000 : NEXT I
270 REM ===============
280 REM LET THE USER SELECT THE NUMBER OF PROBLEMS
290 REM WITHIN ARBITRARY LIMITS OF AT LEAST 3
300 REM BUT NO MORE THAN 15. THIS DEFINES THE QUESTION LOOP.
310 REM ===============
320 PRINT "♡" : FOR I = 1 TO 11 : PRINT : NEXT I
330 PRINT "HOW MANY PROBLEMS DO YOU WANT (3-15)";
340 INPUT P
350 REM ===CHECK TO BE WITHIN RANGE===
360 IF P < 3 OR P > 15 THEN 320
370 REM ===============
380 REM BEGIN THE QUESTION LOOP
390 REM ===============
400 FOR Q = 1 TO P
410 PRINT "♡" : FOR I = 1 TO 11 : PRINT : NEXT I
420 REM ===GENERATE THE TWO RANDOM NUMBERS===
430 N1 = INT(16 * RND(0) + 5)
440 N2 = INT(16 * RND(0) + 5)
500 REM ===SET "A" EQUAL TO THE ANSWER===
510 A = N1 + N2
520 REM ===PRINT OUT THE PROBLEM===
530 PRINT TAB(12) N1 "+" N2 "= ";
540 INPUT R
550 IF A = R THEN 690
560 PRINT : PRINT TAB(7)"NO, THE CORRECT ANSWER IS";A
570 REM ===============
580 REM LET THE USER DETERMINE WHEN TO CONTINUE
590 REM IN THAT THE QUESTION WAS MISSED AND MIGHT
600 REM NEED TO BE "STUDIED."
610 REM ===============
620 PRINT : INPUT " PRESS THE RETURN KEY TO CONTINUE";Z$
630 GOTO 710
640 REM ===============
650 REM FOR EVERY PROBLEM ANSWERED CORRECTLY:
860 REM CLEAR SCREEN, CENTER REINFORCER, HOLD ABOUT 1 SECOND,
670 REM INCREASE CORRECT COUNTER BY 1, AND CONTINUE LOOP.
680 REM ===============
690 PRINT "♡" :FOR I = 1 TO 11:PRINT:NEXT I:PRINT
    TAB(17)"O.K.!"
700 FOR I = 1 TO 1000 : NEXT I : C = C + 1
710 NEXT Q
720 REM ==============
730 REM THIS IS THE END OF THE QUESTION LOOP, SO LET'S
740 REM TELL HOW MANY CORRECT ANSWERS WERE GIVEN.
750 REM =============
760 PRINT "♡" : FOR I = 1 TO 11 : PRINT : NEXT I
770 PRINT "YOU ANSWERED" C "QUESTION(S) CORRECTLY!"
780 END
```

3.5 Posers and Problems

1. Correct any errors (if, in fact, there are any) in the following three programs:

```
10 FOR X = 1 TO 10
20    PRINT Y,Y*Y
30 NEXT Y
40 END
```

```
10 DATA "How old is George",45," (in 1983)"
20 READ Q$,A,C
30 PRINT Q$;C;
40 INPUT R
50 IF A$ = R THEN 70
60 PRINT "No . . .!"
70 PRINT A " is correct!"
80 END
```

```
10 DATA 4,5,6
20 FOR X = 1 TO 3
30    READ A
40    PRINT, A
50 NEXT X
60 READ A
70 PRINT A
80 END
```

2. Below are some student data for a name and test score. Complete the program so that the name and score are printed in columnar form.

```
10 DATA "CHUCK",95,"MARY",80,"PHIL",95,"JEANNIE",35
20 FOR I = 1 TO 4
30   READ S$,S
??
```

3. Modify your program in Problem 2 to print the average of the scores after printing the list of names and scores.

4. Write a search program in which a list of data elements consists of hypothetical names, hair color, and eye color. A list is to be printed that shows the above information based upon a search of a user-specified eye color. For example, if BLUE is input in response to EYE COLOR?, the name, hair color, and eye color for all blue-eyed people would be printed. The program should also give the option to conduct another search.

   Note: This program may be easily completed by modifying PROGRAM 4. For example, PRINT statements describing the program and prompting for the eye color will have to be modified and/or added. The INPUT will have to be a string variable. The READ statement will need to READ three DATA elements. The IF-THEN check will have to compare the INPUT variable and the eye color variable just read. The PRINT statement showing a match will have to be written so that three variable values are displayed on one line. Finally, the DATA statements will have to contain three elements (name, hair color, and eye color).
5. Enter and RUN the following program. Be prepared to discuss its flow.

```
10 FOR R = 10 TO 1 STEP -1
20    PRINT "♡" : PRINT : PRINT
30    PRINT R " little rabbit(s) . . . see the tail(s)!"
40    PRINT,
50      FOR T = 1 TO R
60         PRINT "* ";
70      NEXT T
80    PRINT
90    PRINT : PRINT : PRINT,
100   INPUT "Please press the "RETURN" key. . ."; Z$
110 NEXT R
120 PRINT "CLR/HOME key" : PRINT : PRINT : PRINT
130 PRINT "And then there were NONE . . ."
```

   In particular, what is the purpose of "STEP −1" in statement 10, the commas in statements 40 and 90, the semicolon in statement 60, and the PRINT in statement 80? After RUNning the program, delete statements 20, 90–100, and 120. RUN the program again and note the results.
6. Write a program that converts Celsius to Fahrenheit and PRINTs an equivalence table for every five Celsius degrees from 0 to 100, inclusive. *Hint:* $F = 32 + (C * 9/5)$
7. Write a program that will PRINT the cube of the numbers 1–10, inclusive.
8. Modify PROGRAM 5 to present a drill on multiplication.

9. Modify PROGRAM 5 to present a drill on division using a range of numbers between 4 and 144. Use random numbers that will *not* have a remainder after division. (*Note:* This task may be accomplished by incorporating the statement:

```
490 IF N1/N2 <> INT (N1/N2) THEN 430
```

This will ensure that any quotient derived from dividing N1 by N2 will be a whole number; that is, an integer value with no remainder.)

DIM It! There Must Be An Easier Way! Array! Array! There Is!

"Frustration is not having anyone to blame but yourself."
Bits and Pieces

"Work spares us from three great evils: boredom, vice and need."
Voltaire

Think About This (for Fun)
I have two U.S. coins that total $0.55. One of them is not a nickel. What are the coins?

Think About This (Seriously)
Could the proliferation of microcomputers in the school <u>and</u> home alter the traditional classroom setting as we now know it? If so, how?

4.1 Objectives

For the successful completion of this chapter, you should be able to:

1. Define and give at least one example of a subscripted variable (Section 4.2.1).
2. Define and give at least one example of both a one- and two-dimensional array (Sections 4.2.1 and 4.2.2).
3. Define and give at least one example of each of the BASIC statements DIM and GOSUB-RETURN (Section 4.4).
4. Give examples that indicate how *commands* may be incorporated as *statements* in a BASIC program (Section 4.4.4).
5. Design, enter, and RUN a BASIC program of your own choosing that includes a one-dimensional array and the BASIC statements for this chapter.

4.2 Arrays

4.2.1 One-Dimensional Arrays

For our purposes, a one-dimensional array is just an organized list of information. That information could be any string and/or numeric values: student names, states, chemical names, school districts, test scores, ages, weights, years, and so on. Recall the four names and test scores given in Problem 2 of Chapter 3. Using one-dimensional arrays in BASIC, we can easily make lists of those names and scores.

Consider the BASIC statements needed. First, there is information (names and scores) that will be used; thus, there will be a need for DATA statements. Of course, if there is DATA, it will need to be READ. Also, a FOR-NEXT loop could be used to do the READing. Thus:

```
10 DATA "Chuck","Mary","Phil","Jeannie"
20 FOR I = 1 TO 4
30    READ N$(I)
40 NEXT I
```

This should appear somewhat familiar, with the exception of statement 30, READ N$(I). N$(I) is an example of another type of variable. This type, however, uses an "internal" variable, (I), to distinguish one value of N$(I) from the others. Remember, the variable I is going to have a value that may be 1, 2, 3, or 4 (FOR I = 1 TO 4). Thus, the variable values in this example are:

```
N$(1) = Chuck
N$(2) = Mary
```

```
N$(3) = Phil
N$(4) = Jeannie
```

Or, said another way, there is a four-item list (one-dimensional array) of N$(I) values:

I	N$(I)
1	Chuck
2	Mary
3	Phil
4	Jeannie

The name given to these types of variables is *subscripted variables.* The value of N$(1), pronounced "N$ sub 1," is equal to the string Chuck; the value of N$(2) is equal to the string Mary, and so on.

It is quite simple to build a series of lists using subscripted variables. Consider:

```
10 DATA "Chuck",95,"Mary",80,"Phil",95,"Jeannie",35
20 FOR I = 1 TO 4
30    READ N$(I),S(I)
40 NEXT I
```

If these statements were to be executed, what would be the value of N$(3)? Of S(4)? If the following statements were added to those above, what would be the result of execution?

```
50 FOR I = 4 TO 1 STEP -1
60    PRINT N$(I),S(I)
70 NEXT I
80 END
```

(Mentally trace the execution; then enter and RUN to check your mental interpretation.)

Enter and RUN the following program:

```
10 PRINT "♡" : PRINT : PRINT : PRINT,
20 PRINT "Here's a list of 5 names:" : PRINT
30 FOR I = 1 TO 5 : READ N$(I) : PRINT N$(I) : NEXT I
40 PRINT : PRINT "Let's randomly choose 4 sets of 2 names"
44 INPUT "(PRESS RETURN...)";Z$ : PRINT
50 FOR I = 1 TO 4 : PRINT,
60    FOR J = 1 TO 2
70       X = INT(5 *RND(0) +1)
80       PRINT N$(X),
90    NEXT J
```

```
94 PRINT
100 NEXT I
110 PRINT : PRINT "And here's the list in reverse" : PRINT
120 FOR I = 5 TO 1 STEP -1 : PRINT N$(I) : NEXT I
130 DATA "Kay","Buck","Karyn","Tracy","George"
```

As you can see (and will see again), one advantage of using one-dimensional arrays is that, once the values have been assigned to positions in the list, we may pick and choose items from the list at will.

4.2.2 Two-Dimensional Arrays

A one-dimensional array is nothing more than a *list* of data. A two-dimensional array is a *table* of data in rows and columns.

Assume there are two test scores for each of those students above. A table, consisting of four rows and two columns, might look like this:

Name	Test 1	Test 2
Chuck	95	80
Mary	80	82
Phil	95	93
Jeannie	35	98

Note: The table in this example is composed of the test scores. The names of the students cannot be included in the table because they are *string* variables, and the two-dimensional array is defined as a *numeric* array. In other words, variable types (string and numeric) cannot be mixed in an array. However, just as it is possible to define a two-dimensional numeric array, a two-dimensional string array may also be defined. Just don't mix them!

How could a two-dimensional array of this information be formed? Again, there is information (names and scores) that will be used, so DATA statements are appropriate. This information in statement form would be:

```
10 DATA "Chuck",95,80
20 DATA "Mary",80,82
30 DATA "Phil",95,93
40 DATA "Jeannie",35,98
```

(The DATA statements above could be combined into one or two statements. However, they are listed as four separate statements for the sake of clarity.)

The difficulty is in determining how the DATA should be READ. Examine the sequence of information: one name, followed by two scores, then the next name, followed by two scores; and so on. Since there are four rows (names) and each must be READ, the first loop to be defined will be FOR I = 1 TO 4. However, before the next name is READ, there are two scores to be assigned (READ). So, another loop, FOR J = 1 TO 2, needs to be defined. Thus:

```
 50 FOR I = 1 TO 4
 60    READ N$(I)
 70       FOR J = 1 TO 2
 80          READ S(I,J)
 90       NEXT J
100 NEXT I
```

(*Note:* The indentation is for clarity only.)

The variable I is READing the rows of the table and the variable J is READing the columns. The J loop is said to be *nested* within the I loop. Examine the table above. What is the value of S(I,J) when I = 1 and J = 1? When I = 3 and J = 2? The most important point to remember is that I and J are nothing more than *numbers* that define a row and column, respectively.

A more productive use of two-dimensional arrays may be illustrated by adding the following statements to the program:

```
45 PRINT "NAME","TEST1","TEST2"
46 PRINT "----","-----","-----"
65 PRINT N$(I),
81 PRINT S(I,J),
82 REM T(I) = CUMULATIVE TOTAL EACH STUDENT'S SCORE
84 T(I) = T(I) + S(I,J)
86 REM T = CUMULATIVE TOTAL OF ALL SCORES
88 T = T + S(I,J)
94 REM SKIP A LINE PRIOR TO PRINT OF NEXT NAME
95 PRINT
110 PRINT
120 FOR I = 1 TO 4
130 PRINT N$(I);"'S AVERAGE IS ";T(I)/2
140 NEXT I
150 PRINT "THE CLASS AVERAGE IS ";T/8
160 END
```

Combine *all* the statements in this section into one program; then enter and RUN it. What is the purpose of the comma at the end of statements 65 and 81? Why is a blank PRINT needed in statement 95? Examine the LIST of the complete program very carefully and mentally follow the execution of each statement.

4.3 Examples of the Use of One-Dimensional Arrays

There are many more applications of one- and two-dimensional arrays in instructional computing than just building lists or tables of names and scores. The following two programs give examples of some of these uses.

4.3.1 PROGRAM 6: Random Sentences, Questions, and Responses

One use of one-dimensional arrays may be seen in Program 6. This program demonstrates using one-dimensional arrays in forming random sentences. One array will contain the subjects, another the verbs, and another the direct objects. Using random numbers, a subject, verb, and direct object will be selected from each list and printed as a sentence. Since each list will contain four items, a total of 64 (4 × 4 × 4) different sentences are possible. A one-dimensional array is also used to randomly select the "question" to be asked; that is, "Identify the (SUBJECT/VERB/DIRECT OBJECT)." Another one-dimensional array will contain the "positive feedback" comments given for a correct answer. This feedback will also be randomly selected from the list. Thus, the use of one-dimensional arrays provides a method of increasing the variety of questions, answers, and feedback in instructional computing programs without extensive programming.

RUN from disk and refer to the listing and run of Program 6.

PROGRAM 6

```
10 REM PROGRAM 6 DESCRIPTION
```

Statements 10–170 give a brief description of the program and define the main variables used.

```
20 REM ===============
30 REM THIS PROGRAM DEMONSTRATES BUILDING LISTS OF
40 REM SUBJECTS, VERBS, AND DIRECT OBJECTS, THEN
50 REM RANDOMLY SELECTING FROM EACH TO FORM A
60 REM 'UNIQUE' SENTENCE.  THE SENTENCE PART TO
70 REM IDENTIFY IS ALSO RANDOMLY SELECTED FROM ANOTHER LIST.
80 REM ===VARIABLE DICTIONARY===
90 REM   A   - A RANDOM NUMBER (3-1)
100 REM  A$  - THE CORRECT ANSWER TO SENTENCE PART IDENTITY
110 REM D$() - A LIST OF 4 DIRECT OBJECTS
120 REM F$() - A LIST OF 3 POSITIVE REINFORCERS
130 REM P$() - A LIST OF THE 3 POSSIBLE SENTENCE PARTS
```

```
140 REM R$    - THE RESPONSE TO A GIVEN QUESTION
150 REM S$() - A LIST OF 4 SUBJECTS
160 REM V$() - A LIST OF 4 VERBS
170 REM X,Y, AND Z ARE RANDOM NUMBERS (4-1)
180 REM =============
190 REM DIMENSION THE SIZE OF EACH ARRAY (LIST)
200 REM =============
210 DIM S$(4),V$(4),D$(4),P$(3),F$(3)
```

Statement 210 DIMensions (see Section 4.4.1) the size of the one-dimensional arrays that will be used.

```
220 REM ===DATA SEQUENCE: SUBJECT, VERB, DIRECT OBJECT===
```

Statements 220–280 define as DATA elements the subjects, verbs, and direct objects and READ them into their respective lists. (Note the use of multiple statements in line 280.)

```
230 DATA "SAM","LIKES","DOG","MARY","LOVES","CAT"
240 DATA "HERB", "SOLD", "BIRD", "LISA, "BOUGHT", "CAR"
250 REM ============
260 REM STORE ELEMENTS IN LISTS (ARRAYS)
270 REM ============
280 FOR I=1 TO 4 : READ S$(I),V$(I),D$(I) : NEXT I
290 REM ===STORE SENTENCE PARTS TO IDENTIFY===
```

Statements 290–320 define as DATA elements the "questions" that will be asked and READ them into a list.

```
300 REM ===(THESE WILL BE OUR 'QUESTIONS')===
310 DATA "SUBJECT","VERB","DIRECT OBJECT"
320 FOR I = 1 TO 3 : READ P$(I) : NEXT I
330 REM ============
```

Statements 330–370 define as DATA elements the "positive feedback" comments given for a correct answer and READ them into a list.

```
340 REM STORE FEEDBACK FOR CORRECT ANSWERS
350 REM ============
360 DATA "M A R V E L O U S","F A N T A S T I C","S W E L L"
370 FOR I = 1 TO 3 : READ F$(I) : NEXT I
380 REM =============
390 REM ARBITRARILY, WE'LL SET THE LOOP TO ASK
400 REM ONLY 5 QUESTIONS, RANDOMLY SELECTING THE PARTS
410 REM =============
```

Statements 420–730 define a "question" loop of five questions.

```
420 FOR Q = 1 TO 5
430 PRINT "♡" : PRINT, "GIVEN THE SENTENCE:
440 A = INT(3 * RND(0) + 1)
```

Statements 440–470 give random values for four variables. "A" defines the position in the list of questions. "X" defines position in the list of subjects, "Y" the verb position, and "Z" the direct object position. Statement 490 PRINTs the sentence composed of these randomly selected parts.

```
450 X = INT(4 * RND(0) + 1)
460 Y = INT(4 * RND(0) + 1)
```

```
470 Z = INT(4 * RND(0) + 1)
480 REM ===PRINT THE SENTENCE COMPOSED OF RANDOM PARTS===
490 PRINT : PRINT,S$(X)" "V$(Y)" THE "D$(Z)"."
500 REM ===ASK FOR IDENTITY OF RANDOMLY SELECTED PART===
510 PRINT : PRINT,"IDENTIFY THE "P$(A)"."
```

The randomly selected part to identify is asked in statement 510.

```
520 PRINT : PRINT,"YOUR ANSWER IS: ";
530 INPUT R$
540 REM =============
550 REM SINCE THE PART TO IDENTIFY WAS RANDOMLY
560 REM SELECTED, WE MUST ASSIGN THE CORRECT ANSWER.
570 REM =============
580 ON A GOTO 590,600,610
590 A$ = S$(X) : GOTO 620
```

The correct answer (either the subject, verb, or direct object) is assigned to A$ in statements 590–610. (Note the use of multiple statements in lines 590 and 600.)

```
600 A$ = V$(Y) : GOTO 620
610 A$ = D$(Z)
620 IF A$ = R$ THEN 720
```

The answer INPUT (at statement 530) is then checked for accuracy with the correct answer at statement 620.

```
630 PRINT : PRINT TAB(5)"THE CORRECT ANSWER IS "A$ : PRINT
```

Statements 630–650 give the response for incorrect answers, hold it on the screen until the RETURN key is pressed, and then continue the question loop.

```
640 PRINT TAB(5)"PRESS RETURN TO CONTINUE"; : INPUT Z$
650 GOTO 730
660 REM =============
670 REM HERE'S WHAT HAPPENS FOR CORRECT ANSWERS:
680 REM CLEAR SCREEN, CENTER RANDOM FEEDBACK,
690 REM HOLD IT THERE A MOMENT OR TWO, THEN
700 REM CONTINUE...(ALL THIS WITH MULTIPLE STATEMENTS)
710 REM =============
720 PRINT "♡":FOR I=1 TO 11:PRINT:NEXT I:PRINT,F$(A):FOR I=1 TO 500:NEXT I
```

Statements 720–730 clear the screen, give a random positive feedback based upon the value of variable A, display it for about one second, and then continue the question loop.

```
730 NEXT Q
740 REM ===END OF LOOP AND PROGRAM===
750 PRINT "♡":FOR I = 1 TO 11:PRINT:NEXT I:PRINT,"THAT'S ALL..." : END
```

```
                         RUN
                         GIVEN THE SENTENCE:


                         LISA LOVES THE CAT.
```

```
          IDENTIFY THE DIRECT OBJECT.


          YOUR ANSWER IS ?LISA



          THE CORRECT ANSWER IS CAT

     PRESS RETURN TO CONTINUE?
          GIVEN THE SENTENCE:


          SAM BOUGHT THE DOG.


          IDENTIFY THE DIRECT OBJECT.


          YOUR ANSWER IS ?DOD




          THE CORRECT ANSWER IS DOG

     PRESS RETURN TO CONTINUE?
          GIVEN THE SENTENCE:


          HERB SOLD THE CAT.


          IDENTIFY THE SUBJECT.



          YOUR ANSWER IS ?HERB
          MARVELOUS
          GIVEN THE SENTENCE:


          MARY LOVES THE DOG.


          IDENTIFY THE VERB.


          YOUR ANSWER IS?
              .
              .
              .
```

4.3.2 PROGRAM 7: Random Selection Without Repetition

Another common use of one-dimensional arrays in instructional computing is in building lists of information for retrieval. These lists contain information, such as names, classifications, and addresses; questions, answers, and hints for a given concept or topic; numerical data for analysis; and so on. The next program illustrates one method of building lists and then randomly selecting items of information from them.

The primary purpose of this program is to demonstrate building one-dimensional string arrays (lists) and then randomly selecting from the lists. A user is given the option of making the length of the list from 3 to 15 elements. The lists, containing names and sex, are then made via INPUT statements. The user then has the option of deciding on the number of names to be randomly selected from the list, with each randomly selected name appearing only once. This randomly selected list and the complete list of all names are printed, and the user has the opportunity to create another list or stop.

RUN from disk and refer to the listing and run of Program 7.

PROGRAM 7

```
10 REM PROGRAM 7 DESCRIPTION
```

Statements 10–240 briefly describe the purpose of the program and the variables used.

```
20 REM ==============
30 REM THIS PROGRAM DEMONSTRATES 'FILLING' A
40 REM ONE-DIMENSIONAL ARRAYS THROUGH INPUT.
50 REM RANDOM SELECTION WITHOUT REPETITION OF
60 REM ANY PREVIOUSLY SELECTED ITEM IS SHOWN.
70 REM THIS IS DONE BY DEFINING A LIST OF "ZEROS"
80 REM AND 'FLAGGING' EACH SELECTED POSITION.
90 REM ===============
100 REM VARIABLE DICTIONARY
110 REM ===============
120 REM N    - THE NUMBER OF NAMES IN A LIST (VIA INPUT)
130 REM N$() - HYPOTHETICAL NAMES
140 REM S    - THE NUMBER OF NAMES RANDOMLY SELECTED FROM THE LIST
150 REM S$() - SEX (M = MALE, F = FEMALE), ALSO VIA INPUT
160 REM X    - A RANDOM NUMBER
170 REM Z(X) - A 'FLAG' FOR THE RANDOM NUMBERS (EITHER 0 OR 1)
180 REM        IF Z(X) = 0 THE NUMBER X HAS NOT BEEN SELECTED
190 REM        IF Z(X) = 1 THE NUMBER X HAS BEEN SELECTED
200 REM ==============
210 REM DIMENSION THE VARIABLES USED TO THEIR MAXIMUM
220 REM SIZE, CLEAR THE SCREEN, AND ASK FOR
```

```
230 REM THE NUMBER OF NAMES TO BE INPUT
240 REM =============
250 DIM N$(15),S$(15),Z(15)
```

Statement 250 DIMensions (see Section 4.4.1) the length of the one-dimensional arrays (lists) to be a maximum of 15 elements.

```
260 PRINT "♡" : FOR I = 1 TO 11:PRINT:NEXT I
270 PRINT "NUMBER OF NAMES TO ENTER (3-15)";
```

Statements 270–290 allow the user to define the number of elements in the list.

```
280 INPUT N
290 IF N < 3 OR N > 15 THEN 250
300 FOR I = 1 TO N
```

Statements 300–420 request a name [assigned to N$(I)] and sex [assigned to S$(I)] for each of the elements. (Note the use of IF-OR-THEN in statement 410.)

```
310 REM ===INITIALIZE 'FLAG' LIST TO ZERO===
320 Z(I) = 0
330 PRINT "♡" : FOR P=1 TO 10 : PRINT : NEXT P
340 PRINT,"NAME NUMBER" I;
350 INPUT N$(I) : PRINT
360 PRINT,"SEX (M OR F)";
370 INPUT S$(I)
390 REM ===============
390 REM CHECK FOR CORRECT ENTRY. NOTE THE USE OF IF-AND-THEN PRINT
400 REM ===============
410 IF S$(I) <> "M" AND S$(I) <> "F" THEN PRINT,"FUNNY SEX?" : GOTO 360
420 NEXT I
430 PRINT "♡" : FOR P = 1 TO 11 : PRINT : NEXT P
```

Statements 430–500 give the user the option of defining the number of random selections to be made from the lists just entered.

```
440 PRINT "YOU HAVE" N "NAMES. HOW MANY DO YOU"
450 PRINT "WANT TO RANDOMLY SELECT";
460 INPUT S
470 REM ===CAN THAT MANY BE SELECTED?===
480 IF S > 0 AND S <= N THEN 510
490 PRINT "THAT IS NOT POSSIBLE. YOU HAVE"
500 PRINT "ENTERED" N "NAMES!" : GOTO 440
510 PRINT "♡"
```

Statements 510–540 print a heading for the lists of randomly selected items.

```
520 PRINT;S "RANDOMLY SELECTED NAME(S)" : PRINT
530 PRINT,"NUMBER","NAME","SEX"
540 PRINT,"------","----","---"
550 REM ===============
560 REM BEGIN SELECTING NAMES AT RANDOM BY THEIR
570 REM POSITION IN THE LIST. PRINT A MESSAGE
580 REM IF THE SAME POSITION IS SELECTED MORE THAN ONCE.
590 REM ===============
600 FOR I = 1 TO S
```

Statements 600–720 print the lists of randomly selected numbers, names, and sexes.

Statement 600 defines the upper limit of the loop to be the value of the variable S (the number of names to be selected, which was INPUT at statement 450).

```
610 REM ===RANDOM VALUE FOR 'X' WITHIN THE LIMITS OF 'N'===
620 X = INT(N * RND(0) + 1)
```

Statement 620 assigns to variable X some random value between 1 and the variable N (the number of names in the complete list, which was INPUT at statement 340).

```
630 REM ===HAS THIS VALUE OF 'X' APPEARED BEFORE?===
640 IF Z(X) = 1 THEN PRINT "THE NUMBER" X "ALREADY CHOSEN!" : GOTO 620
```

Statement 640 checks the value of variable Z(X). If this value is 1 all values of Z(X) were initialized to zero in statement 320, the random number represented by X has appeared previously. If so, a message to that effect is displayed and transfer is back to 620 for another random number.

```
650 REM ===============
660 REM DISPLAY THE NUMBER, NAME, AND SEX OF THE POSITION SELECTED
670 REM AND "FLAG" THAT POSITION IN THE LIST.
680 REM ===============
690 PRINT,X,N$(X),S$(X)
```

Statement 690 prints the values of X, N$(X), and S$(X). Statement 710 "flags" the selection of this value of X by setting Z(X) = 1, and the loop continues at statement 720 until completion.

```
700 REM ===HERE'S THE FLAGGIN'===
710 Z(X) = 1
720 NEXT I
730 PRINT : PRINT
740 INPUT "PRESS RETURN FOR THE COMPLETE LIST";Z$
750 PRINT "♡"
```

After the randomly selected list has been printed, statements 750–810 print the complete lists of names and sexes.

```
760 PRINT,"THE COMPLETE LIST IS:" : PRINT
770 PRINT,"NUMBER","NAME","SEX"
780 PRINT,"------","----","---"
790 FOR I = 1 TO N
800 PRINT,I,N$(I),S$(I)
810 NEXT I
820 PRINT : PRINT
```

Statements 820–880 give the opportunity for the program to be RUN again, creating a new list and randomly selecting from it, or to end the program.

```
830 INPUT "CREATE ANOTHER LIST (Y OR N)";Z$
840 IF Z$ = "Y" THEN 290
850 PRINT "♡" : FOR P = 1 TO 11 : PRINT : NEXT P
860 PRINT TAB(15) "D O N E"
870 FOR P = 1 TO 2000 : NEXT P : PRINT "♡"
880 END
```

```
RUN
NUMBER OF NAMES TO ENTER (3-15)?7
NAME NUMBER 1?FRANK
SEX (M OR F)?M
NAME NUMBER 2?FRAN
SEX (M OR F)?F
NAME NUMBER 3?BILL
SEX (M OR F)?M
NAME NUMBER 4?BARB
SEX (M OR F)?F
NAME NUMBER 5?BUCK
SEX (M OR F)?B
                THAT'S A FUNNY SEX!
SEX (M OR F)?M
NAME NUMBER 6?KAY
SEX (M OR F)?F
NAME NUMBER 7?TRACY
SEX (M OR F)?F
YOU HAVE 7 NAMES. HOW MANY DO YOU
WANT TO RANDOMLY SELECT12
THAT IS NOT POSSIBLE. YOU HAVE
ENTERED 7 NAMES!
YOU HAVE 7 NAMES. HOW MANY DO YOU
WANT TO RANDOMLY SELECT5

5 RANDOMLY SELECTED NAMES:
NUMBER          NAME          SEX
------          ----          ---
6               KAY           F
7               TRACY         F
THE NUMBER 6 HAS BEEN SELECTED BEFORE
4               BARB          F
3               BILL          M
2               FRAN          F

THE COMPLETE LIST IS:
NUMBER          NAME          SEX
------          ----          ---
1               FRANK         M
2               FRAN          F
3               BILL          M
4               BARB          F
5               BUCK          M
6               KAY           F
7               TRACY         F
```

```
CREATE ANOTHER LIST (Y OR N)N


                    D O N E
```

4.3.3 Use of a "Flag" for Nonrepetitive Random Selection

In Program 7, all values for Z(X) are initialized to zero (statement 320) as the name and sex information is INPUT. A partial list of the information INPUT in the sample RUN of Program 7 would be:

X	N$(X)	S$(X)	Z(X)
1	Frank	M	0
2	Fran	F	0
3	Bill	M	0
4	Barb	F	0
:	:	:	:
:	:	:	:

If, for example, a random value for X is 2 (statement 620), then Z(2) is set to 1, that is, "flagged" (statement 710), and the list would now be:

X	N$(X)	S$(X)	Z(X)
1	Frank	M	0
2	Fran	F	1
3	Bill	M	0
4	Barb	F	0
:	:	:	:
:	:	:	:

Since Z(2) is now equal to 1, any subsequent random value where X is equal to 2 would cause statement 640 to transfer execution *back* to statement 620, where another random value for X would be generated. Therefore, any randomly selected name in this program example will be displayed only one time.

4.4 BASIC Statements for This Chapter

4.4.1 Statement—DIM

Purpose

DIMensions (defines the size needed) for one- and two-dimensional arrays. On most BASIC systems, it is unnecessary to use the DIM statement unless the array will contain more than ten elements. Most systems automatically allocate space for ten or less elements. However, it is good programming practice to DIMension all arrays, even those containing ten or less elements.

Examples:

```
DIM N$(12),Z(5)
```

(Dimensions space for a list of 12 string variables and a list of 5 numeric variables.)

```
DIM S(20,3)
```

(Dimensions space for a 20-row, 3-column table containing numeric data.)

4.4.2 Statement Pair—GOSUB-RETURN

Purpose

This statement pair is very useful for programs in which a sequence of statements is repeated several times throughout program execution. Whenever GOSUB is encountered, program execution is transferred to the statement number specified in the GOSUB. Execution continues from that statement until the RETURN statement is encountered. Execution is then transferred (RETURNed) to the statement number *immediately following* the GOSUB statement that caused the transfer in the first place.

A typical example in instructional computing would be an answer-checking sequence for student input in a program containing several questions. Rather than writing an identical answer-checking sequence for each question, it is written only once as a subroutine. GOSUB may then be used after each question to check the answer.

Example:

```
10 PRINT "CAPITAL OF TEXAS";
20 A$ =  "AUSTIN"
30 GOSUB 1000
40 PRINT "X MARKS THE --?--";
50 A$ =  "SPOT"
60 GOSUB 1000
980 PRINT, "BYE-BYE . . ."
990 END
1000 INPUT R$
1010 IF R$ = A$ THEN PRINT "GREAT!" : RETURN
1020 PRINT "NOPE . . .IT'S " A$
1030 RETURN
```

Mentally execute this program before entering and RUNning it. Note that the program has room to incorporate other questions between statements 60–980, that RETURN may be with different statements in the subroutine, and that END may come before the subroutine.

4.4.3 PROGRAM 8: Question Sets and Hints Using GOSUB-RETURN Statements

The following program illustrates the use of GOSUB-RETURN in a "muscle quiz." Note that alternate correct answers are used, note how the questions, hints, and answers are assigned and presented, and note that credit is given only for those answers that are correct on the first try. Mentally outline the execution of this program carefully. Program 8 will not be discussed as the previous programs were.

RUN from disk and refer to the listing and run of Program 8.

PROGRAM 8

```
10 REM  ===============
20 REM   PROGRAM 8 DESCRIPTION
30 REM  ===============
40 REM  PROGRAM DEMOS THE USE OF
50 REM  GOSUB-RETURN, CHECKING FOR
60 REM  SYNONYMOUS CORRECT ANSWERS
70 REM  AND ASSIGNMENT OF HINTS.
80 REM  ===============
90 REM  VARIABLE DICTIONARY
100 REM  ===============
110 REM  C  - CORRECT ANSWER COUNTER
120 REM  C$ - SYNON. CORRECT ANSWER
130 REM  D$ - SYNON. CORRECT ANSWER
140 REM  E$ - SYNON. CORRECT ANSWER
150 REM  F  - FLAG FOR MISS ON 1ST TRY
160 REM  H$ - THE HINT GIVEN ON 1ST MISS
170 REM   ===============
180 PRINT "♡"
190 PRINT "    M U S C L E   Q U I Z"
200 PRINT
210 REM  ===============
220 REM  PRINT THE QUESTION, DEFINE
230 REM  THE CORRECT ANSWER(S) AND THE
240 REM  HINT TO BE GIVEN, THEN USE THE
250 REM  SUBROUTINE FOR INPUT AND CHECKING
260 REM  ===============
270 REM
280 PRINT "WHAT IS THE LARGEST MUSCLE"
290 PRINT "   IN THE HUMAN BODY ";
300 C$ = "GLUTEUS MAXIMUS"
```

```
310 D$ = "BUTTOCKS"
320 E$ = "DERRIERE"
330 H$ = "CHANCES ARE IT'S FLATTENED RIGHT NOW"
340 GOSUB 5000
350 REM  ===============
360 REM  REPEAT THE QUESTION AND
370 REM  ANSWER SEQUENCE AGAIN
390 REM  ===============
390 PRINT "WHAT MUSCLE IS CONSIDERED BY SOME"
400 PRINT "   TO HAVE AN ORIGIN"
410 PRINT "BUT NO INSERTION ";
420 REM  ===ONLY ONE ANSWER===
430 C$ = "TONGUE"
440 D$ = C$
450 E$ = C$
460 H$ = "ON SOME, IT 'WAGS' A LOT!"
470 GOSUB 5000
480 REM  ===============
480 REM  REPEAT THE PROCESS
500 REM  FOR A FINAL QUESTION
510 REM  ===============
520 PRINT "WHAT MUSCLE HAS MADE MARK EDEN RICH,"
530 PRINT "   OLD MEN LEER"
540 PRINT "AND WEIGHT LIFTERS STRUT ";
550 C$ = "PECTORAL"
560 D$ = "PECTORALIS MAJORA"
570 E$ = "PECTORALS"
590 H$ = "PALM TO PALM PRESSURE DEVELOPS IT!"
590 GOSUB 5000
600 REM  ===============
610 REM  THERE'S PLENTY OF ROOM
620 REM  TO ADD MORE QUESTIONS AND
630 REM  ANSWERS FOLLOWING THE SAME
640 REM  SEQUENCE AS ABOVE
650 REM  WITH PRINTS, SYNONYMOUS ANSWER
660 REM  AND HINT ASSIGNMENTS FOLLOWED
670 REM  BY A GOSUB 5000
680 REM  ===============
690 REM  ===============
700 REM  STATEMENT 4990 IS NEEDED
710 REM  TO SKIP THE SUBROUTINE,
720 REM  GIVE THE PERFORMANCE SCORE
730 REM  AND END THE PROGRAM
740 REM  ===============
4990 GOTO 5190
4994 REM  ===============
4996 REM  SUBROUTINE PORTION OF PROGRAM
4998 REM  ===============
5000 INPUT A$ : PRINT
5010 IF A$ = C$ OR A$ = D$ OR A$ = E$ THEN 5080
5020 REM ===GIVE A HINT ON THE FIRST MISS===
5030 IF F = 0 THEN F = 1 : PRINT H$ : PRINT "NEXT ANSWER"; : GOTO 5000
5040 PRINT: PRINT, "CORRECT ANSWERS ARE:"
5050 PRINT : PRINT C$,D$,E$
5060 PRINT : INPUT "PRESS RETURN TO CONTINUE...";Z$ : GOTO 5120
5070 REM ===NO CREDIT IS GIVEN FOR CORRECT ANSWERS ON 2ND TRY===
5080 IF F = 1 THEN 5100
5090 C = C + 1
5100 PRINT:PRINT:PRINT TAB(15)"O.K.!" : FOR P = 1 TO 1000 : NEXT P
5110 REM  ===SET F TO ZERO BEFORE NEXT QUESTION===
5120 F = 0
5130 PRINT "♡"
5140 RETURN
5150 REM  ===============
5160 REM  END OF SUBROUTINE
5170 REM  ===============
5180 PRINT
5190 PRINT "YOU GOT" C "CORRECT ON THE FIRST TRY."
5200 PRINT : PRINT ,"THAT'S ALL..."
5210 END
```

```
RUN
   M U S C L E   Q U I Z

WHAT IS THE LARGEST MUSCLE
   IN THE HUMAN BODY ?BUTTOCKS
O.K.

WHAT MUSCLE IS CONSIDERED BY SOME
   TO HAVE AN ORIGIN
BUT NO INSERTION ?BICEPS
ON SOME, IT 'WAGS' A LOT.
YOUR NEXT ANSWER IS?TAIL

                CORRECT ANSWERS ARE:
TONGUE          TONGUE          TONGUE

PRESS RETURN TO CONTINUE...

WHAT MUSCLE HAS MADE MARK EDEN RICH,
   OLD MEN LEER
AND WEIGHT LIFTERS STRUT ?CHEST
PALM TO PALM PRESSURE DEVELOPS IT!
YOUR NEXT ANSWER IS?PECTORALS
O.K.
```

```
YOU GOT 1 CORRECT ON THE FIRST TRY.

        THAT'S ALL...
```

4.4.4 Incorporating Commands in a Program

By now, you are well aware of several of the BASIC commands (LOAD, RUN, LIST, etc.) of the Commodore 64 microcomputer. Many of these commands may be incorporated within the body of a BASIC program. When these are executed they could, for example, execute (RUN) another program from the disk. This example is particularly useful in that one program can "command" another program to RUN, which could command another program to RUN, and so on. Thus, programs can be "linked" together in "chain" fashion, rather than writing, entering, testing *one long* program.

A common use in instructional computing is to design one program as a "menu" of available programs on the disk. When this *menu program* is executed, a selection of programs is displayed, and the user may enter the choice desired. Based upon the user's input, execution is transferred to the appropriate line number in the menu program that commands the system to RUN the program selected. When certain statements in our example menu program are executed, the system will automatically LOAD and RUN the program named Sequel 1. At the conclusion of Sequel 1, these statements could be incorporated to LOAD and RUN the program named Sequel 2, and so on.

This use of commands as BASIC statements is illustrated in the program named MENU on the disk. Carefully examine the listings of MENU, SEQUEL 1, and SEQUEL 2; then, RUN MENU and note the options and actions it provides. In particular, note the method in MENU by which program names are assigned to a one-dimensional array. Then note how the selected program is RUN. Compare this use of RUN as a statement with those in SEQUEL 1 and SEQUEL 2.

Menu

```
10 REM  MENU DESCRIPTION
20 REM  ================
30 REM  PROGRAM DEMONSTRATES HOW A 'MENU' OF PROGRAMS
40 REM  MAY BE PROVIDED TO A USER.  USING 'POKE'
50 REM  STATEMENTS, ONE PROGRAM WILL AUTOMATICALLY
60 REM  'LOAD' AND 'RUN' THE PROGRAM DEFINED AS
70 REM  THE STRING P$(C).
80 REM  ================
90 REM
100 PRINT "♡" : PRINT : PRINT
```

```
110 PRINT TAB(6) "P R O G R A M   M E N U" : PRINT
120 PRINT,"YOUR OPTIONS ARE:" : PRINT
130 PRINT,"1. SEQUEL 1"
140 PRINT,"2. SEQUEL 2"
150 PRINT,"3. STOP"
160 PRINT : PRINT, : INPUT "YOUR CHOICE IS (1-3):";C
164 IF C < 1 OR C > 3 THEN 160
170 IF C = 3 THEN PRINT "♡" : PRINT "SELECTIONS COMPLETED" : END
180 REM
190 REM ===STORE VALUES IN P$()===
200 REM
210 FOR I = 1 TO 2 : READ P$(I) : NEXT I
220 PRINT "♡"
230 REM
240 REM ===LOAD AND RUN USING 'POKES'===
250 REM
260 PRINT "LOAD ";CHR$(34);P$(C);CHR$(34);",8"
270 POKE 631,19 : POKE 632,17 : POKE 633,13
280 POKE 634,82 : POKE 635,85 : POKE 636,78 : POKE 637,13 : POKE 198,7
290 REM
300 REM ===DATA FOR PROGRAM NAMES TO LOAD AND RUN===
310 REM
320 DATA "SEQUEL 1","SEQUEL 2"
```

Sequel 1

```
10 REM  SEQUEL 1 DESCRIPTION
20 REM  ===============
30 REM  ONE OF THE PROGRAMS ACCESSED VIA 'MENU' PROGRAM.
40 REM  ===============
50 REM
60 PRINT "♡" : FOR I = 1 TO 10 : PRINT : NEXT I
70 PRINT "AND HERE WE ARE EXECUTING 'SEQUEL 1'..." : PRINT
80 PRINT "IF YOU PRESS THE LETTER 'N' (FOR NEXT),"
90 PRINT "WE'LL GO TO 'SEQUEL 2'.  ANY OTHER KEY"
100 PRINT "WILL TAKE YOU BACK TO 'MENU'." : PRINT
110 PRINT "WHAT'S YOUR PLEASURE?";
120 GET Z$ : IF Z$ = "" THEN 120
130 IF Z$ = "N" THEN F$ = "SEQUEL 2" : GOTO 150
140 F$ = "MENU"
150 PRINT "♡"
160 REM
170 REM ===USE OF POKES TO LOAD AND RUN 'F$'===
180 PRINT "LOAD ";CHR$(34);F$;CHR$(34);",8"
190 POKE 631,19 : POKE 632,17 : POKE 633,13
200 POKE 634,82 : POKE 635,85 : POKE 636,78 : POKE 637,13 : POKE 198,7
```

Sequel 2

```
10 REM  SEQUEL 2 DESCRIPTION
20 REM  ===============
30 REM  ONE OF THE PROGRAMS ACCESSED VIA 'MENU' PROGRAM.
40 REM  ===============
50 REM
60 PRINT "♡" : FOR I = 1 TO 10 : PRINT : NEXT I
70 PRINT "AND HERE WE ARE EXECUTING 'SEQUEL 2' ..." : PRINT
90 PRINT "NOW, WHEN YOU PRESS ANY KEY, I"
100 PRINT "WILL TAKE YOU BACK TO 'MENU'," : PRINT
110 PRINT "I'M WAITING..."
120 GET Z$ : IF Z$ = "" THEN 120
140 F$ = "MENU"
150 PRINT "♡"
160 REM
170 REM ===USE OF POKES TO LOAD AND RUN 'F$'===
180 PRINT "LOAD" ;CHR$(34) ;F$ ;CHR4(34) ;",8"
190 POKE 631,19 : POKE 632,17 : POKE 633,13
200 POKE 634,82 : POKE 635,85 : POKE 636,78 : POKE 637,13 : POKE 1

RUN


          P R O G R A M   M E N U


          YOUR OPTIONS ARE:
          ---- --------
            1. SEQUEL 1
            2. SEQUEL 2
            3. STOP
          YOUR CHOICE (1-3)?1


AND HERE WE ARE EXECUTING
PROGRAM 'SEQUEL 1'...


IF YOU DEPRESS THE LETTER 'N'
(FOR 'NEXT'), WE'LL GO TO
'SEQUEL 2'. ANY OTHER KEY
WILL TAKE YOU BACK TO THE 'MENU'...

WHAT'S YOUR PLEASURE? N


AND HERE WE ARE EXECUTING 'SEQUEL 2'...

NOW, WHEN YOU PRESS ANY KEY, I
WILL TAKE YOU BACK TO 'MENU'.

I'M WAITING...
```

```
          P R O G R A M   M E N U

          YOUR OPTIONS ARE:
          ---- --------
            1. SEQUEL 1
            2. SEQUEL 2
            3. STOP
          YOUR CHOICE (1-3)?3

     SELECTIONS COMPLETED
```

4.5 Posers and Problems

1. Assume you have a class of 20 students and the semester test average and final exam scores for each. Outline the BASIC statements that would make a series of lists of this information.
2. Outline the BASIC statements that will read 3 scores for each of 25 students into a two-dimensional array.
3. Write a brief paragraph outlining the execution of the program consisting of the combined statements in Section 4.2.2.
4. Describe what would result from execution of the following BASIC statements:

```
10 DATA "TEXAS","OKLAHOMA","KANSAS","NEVADA","UTAH"
20 FOR I = 1 TO 5
30    READ S$(I)
40 NEXT I
50 FOR K = 1 TO 3
60    X = INT(5 * RND(0) + 1
70    PRINT S$(X)
80 NEXT K
90 END
```

5. How would adding the following statements affect the execution of the program in Problem 4?

```
34 Z(I) = 0
64 IF Z(X) = 1 THEN 60
66 Z(X) = 1
```

6. Describe the execution of the program in Problem 5 if statement 50 were changed to:

```
50 FOR K = 1 TO 6
```

   Think this through carefully before entering and RUNning. Remember, depressing the RUN/STOP and RESTORE keys simultaneously will halt a runaway program!

7. How would you modify the program in Section 4.4.2 to print a hint as a response for the first miss and give the correct answer on the second miss? (*Hint:* One way to do this is shown in Program 8.)
8. Write a program that creates five random sentences from lists of subjects, verbs, and objects.
9. Write a program to choose and print four random numbers between 1 and 10 without repeating any number that has been printed.
10. Write a program that will be a Drill Menu for the addition, subtraction, multiplication, and division programs written in Chapter 3. Modify each of the drill programs so that at their conclusions the Drill Menu program will be RUN. *NOTE!!!* SAVE your Drill Menu program before RUNning to test your design. (Why is this necessary?)

Chapter Five

Relax and Catch Your BASIC Breath

*" 'Tis an old tale, and
often told."*
Walter Scott

*"For time will teach thee
soon the truth /
There are no birds
in last year's nest."*
H.W. Longfellow

*"And look before you,
ere you leap /
For as you sow,
y' are like to reap."*
Samuel Butler

Think About This (for Fun)
Four men are on a raft in the middle of the ocean. Each has one carton of cigarettes but no means whatsoever of lighting them. The smartest of the four, however, devotes his full mental prowess to the problem and, within minutes, all are smoking a cigarette. How was this accomplished?

Think About This (Seriously)
Could the role of the teacher change as computers become common in our schools? If so, how do you see this new role?

5.1 Objectives

For the successful completion of this chapter, you should be able to:

Design, enter, and RUN a minimum of three short programs using the BASIC statements summarized below.

5.2 BASIC Statements: A Summary and Some Typical Uses

As an examination of any BASIC text or manual will show, there is more to the language than has been discussed so far. To this point, nothing more has been presented than a brief introduction to the basics of BASIC. Although there is *much more* (some of which will be presented in later chapters), the statements and commands discussed to this point are the foundation for instructional computing applications.

This chapter summarizes these statements and broadly outlines their use in designing application programs. This summary must be general in illustrating applications since, as in any writing endeavor (programming or otherwise), the author's creativity is the limiting factor. The BASIC statements are only the *means* by which programs can be constructed. The content, design strategies, effectiveness, and applicability of programs are the end result of creativity. Programs can only be as good—or as bad—as this factor.

Note: Many of the following examples are actually program fragments that may be referenced for future program development.

5.2.1 PRINT

This statement displays (outputs) information.

Some typical uses	Example
Output text	`PRINT "WHAT'S YOUR NAME";`
Text + numeric var	`PRINT "SCORE IS " S " PERCENT"`
Text + string var	`PRINT "HELLO, " N$ "!"`
Numeric with spacing	`PRINT A,B,C`
String (close packed)	`PRINT N$;S$;A$`

5.2.2 LET (Optional)

This statement allows the assignment of a value to a variable.

Some typical uses	Example
Counters	`C = C + 1`
Assign correct answer	`A$ = "AUSTIN"`
	`A = Y * Z`
Assigning hints	`H$="'RIVER CITY'"`
Computation	`S = C * 100/Q`

5.2.3 INPUT

This statement receives information (input) from the keyboard and assigns it to a defined variable.

Some typical uses	Example
Numeric input	`INPUT N`
String input	`INPUT N$`
Combinations	`INPUT N$,N`
	`INPUT X,Y,Z`

5.2.4 GOTO

This statement acts as an unconditional branch (transfer of execution) to a specified statement number.

Some typical uses	Example
Skipping statements	`100 GOTO 130`
	`110 PRINT "VERY GOOD!"`
	`120 C = C + 1`
	`130 PRINT "NEXT QUESTION . . ."`
	`* * * * *`
Returning for input	`100 PRINT : PRINT TAB(12);`
	`110 INPUT "YOUR ANSWER IS";R`
	`.`
	`.`
	`.`
	`150 PRINT "NO, TRY AGAIN . . ."`
	`160 GOTO 100`
	`* * * * *`

(*Note:* The three vertical dots denote omitted program segments.)

5.2.5 IF-THEN

This statement acts as a conditional branch to specified statement number if the defined condition is *true.*

Some typical uses	Example

Answer checking	`100 IF R$=A$ THEN PRINT "GOOD SHOW!"`
	`: GOTO 120`
	`110 PRINT "NO, THE ANSWER IS " A$`
	`.`
	`.`
	`.`

Determining the sequence of execution	`140 IF F = 1 THEN 160`
	`150 C = C + 1`
	`160 PRINT " NEXT QUESTION . . ."`
	`.`
	`.`
	`.`
	`* * * * *`

	`100 IF R < 40 THEN 140`
	`110 IF R > 40 THEN 160`
	`.`
	`.`
	`.`
	`140 PRINT "TOO LOW . . ."`
	`.`
	`.`
	`.`
	`160 PRINT "TOO HIGH . . ."`
	`* * * * *`

	`100 IF F = 1 THEN 200`
	`110 F = 1`
	`120 PRINT "HERE'S A HINT . . ."`
	`.`
	`.`
	`.`
	`200 PRINT "THE CORRECT ANSWER IS " A`
	`* * * * *`

	`100 IF A$ = "STOP" THEN 800`
	`.`
	`.`
	`.`
	`800 PRINT "HERE'S YOUR SCORE . . ."`
	`* * * * *`

Combinations of statements

```
100 IF A$ = R$ THEN C = C + 1
 * * * * *
 .
 .
 .
100 IF A < 1 OR A > 10 THEN
    PRINT "♡"
110 PRINT : PRINT "OUT OF RANGE"
 .
 .
 .
 * * * * *
 .
 .
 .
800 PRINT "YOUR GRADE IS ";
810 IF S > 99 THEN PRINT "A"
820 IF S > 79 AND S < 90 THEN PRINT
    "B"
 .
 .
 .
 * * * * *
```

5.2.6 ON-GOTO

This statement acts as a branch to a specified statement based on the value of a defined variable or expression.

Some typical uses | **Example**

Branch to a randomly selected question

```
 .
 .
 .
100 X = INT(5 * RND(0) +1)
 .
 .
 .
150 ON X GOTO 200,300,400,500,600
 .
 .
 .
200 PRINT "QUESTION 1 . . ."
 .
 .
 .
300 PRINT "QUESTION 2 . . ."
 .
 .
 .
```

Branch to a hint for a given question	
	```
	* * * * *
	.
	.
	.
	100 PRINT "QUESTION 1 . . ."
	110 Q = 1
	.
	.
	.
	500 INPUT "YOUR ANSWER IS";R$
	.
	.
	.
	550 PRINT "HERE'S A HINT . . ."
	560 ON Q GOTO 700,800,900
	.
	.
	.
	700 PRINT "HINT FOR QUESTION 1 . . ."
	710 GOTO 500
	.
	.
	.
	* * * * *
	```

5.2.7 DATA-READ (Statement Pair)

This statement "stores" and assigns (READs) information to a defined variable.

Some typical uses	Example

Assignment of question answer and hint	
	```
	100 REM *** Q$=QUES A$=ANS H$=HINT
	110 DATA "TEXAS","AUSTIN","RIVER
	    CITY"
	.
	.
	.
	200 READ Q$,A$,H$
	.
	.
	.
	* * * * *
	```

Assignment of numerical information	
	```
	100 DATA 90,76,55,70,88,93
	.
	.
	.
	150 READ S1,S2,S3
	160 T = T + S1 + S2 + S3
	.
	.
	.
	* * * * *
	```

5.2.8 FOR-NEXT (Statement Pair)

This statement repeats (loops) statement sequence between the FOR and the NEXT a defined number of times.

Some typical uses | **Example**

Assigning data to arrays

```
200 PRINT "NUMBER OF SCORES";
210 INPUT N
220 FOR I = 1 TO N
230    PRINT "SCORE";
240    INPUT S(I)
250    T = T + S(I)
260 NEXT I
270 A = T/N
280 PRINT "AVERAGE SCORE IS " A
 .
 .
 .
 * * * * *

100 DATA "QUES 1","ANS 1","HINT 1"
110 DATA "QUES 2","ANS 2","HINT 2"
 .
 .
 .
200 FOR I = 1 TO 10
210    READ Q$(I),A$(I),H$(I)
220 NEXT I
 .
 .
 .
 * * * * *
```

Checking response to match any defined answer

```
 .
 .
 .
200 FOR I = 1 TO 10
210    IF R$ = A$(I) THEN 240
220 NEXT I
230 GOTO 500
240 PRINT "MATCHES ANS NO. " I
 .
 .
 .
 * * * * *
```

Defining the number of questions to be asked

```
 .
 .
 .
100 PRINT "HOW MANY DO YOU WANT";
110 INPUT P
120 IF > 1 AND P < 11 THEN 150
130 PRINT "THAT'S TOO MANY!"
```

Some typical uses	Example

	`140 GOTO 100`
	`150 FOR I = 1 TO P`
	`160     PRINT "PROBLEM NUMBER " P`
	`.`
	`.`
	`.`
	`300 NEXT I`
	`.`
	`.`
	`.`
	`* * * * *`

5.2.9 GOSUB-RETURN (Statement Pair)

This statement GOes to the statement number defining the SUBroutine, executes the statements in sequence until RETURN is executed, and then returns to the statement following the GOSUB.

Some typical uses	Example

Answer-checking sequence	`.`
	`.`
	`.`
	`100 PRINT "QUESTION 1 . . ."`
	`105 A$ = "ANSWER 1"`
	`110 GOSUB 2000`
	`120 PRINT "QUESTION 2 . . ."`
	`125 A$ = "ANSWER 2"`
	`130 GOSUB 2000`
	`.`
	`.`
	`.`
	`2000 INPUT R$`
	`2010 IF R$ = A$ THEN 2080`
	`.`
	`.`
	`.`
	`2080 C = C + 1`
	`2090 PRINT "HOT-DOGGIES!"`
	`2100 RETURN`
	`* * * *`

Generating random numbers	`.`
	`.`
	`.`
	`100 GOSUB 500`

```
110 PRINT "IF MASS = " X " AND"
120 PRINT "VOLUME = " Y
    ", DENSITY =";
130 INPUT R
140 IF R=Z THEN 200
 .
 .
 .
500 X = INT(1000 * RND(0) + 1)/100
510 Y = INT(500 * RND(0) + 1)/10
520 Z = X/Y
530 RETURN
 * * * * *
```

Generating random responses

```
10 DATA "GOOD","GREAT","WOW"
20 FOR I = 1 TO 3
30 READ R$(I)
40 NEXT I
 .
 .
 .
500 GOSUB 750
 .
 .
 .
750 X = INT(3 * RND(0) + 1)
760 PRINT R$(X) "!"
770 RETURN
 * * * * *
```

Generating messages, Screen positioning

```
100 GOSUB 5000
 .
 .
 .
5000 PRINT "♡" : PRINT,
5010 PRINT "(Message . . .)"
5020 PRINT : PRINT
5030 RETURN
 * * * * *
```

5.3 A Summary of the Purposes of BASIC Statements

Although there are more statements to the BASIC language, the ones summarized in this chapter are fundamental to the construction of instructional computing programs. In essence, these statements form the foundation upon which a program author has:

1. Some means of assigning and/or displaying values.
2. Some means of controlling the screen display and the sequence of execution.
3. Some means of easing repetitious tasks.

Thus, most of the BASIC statements discussed to this point may be further summarized into three categories:

Assignment	Control	Repetition
PRINT	END	FOR-NEXT
LET	PRINT "♡"	GOSUB-RETURN
INPUT	IF-THEN	
DATA-READ	ON-GOTO	
DIM	GOTO	

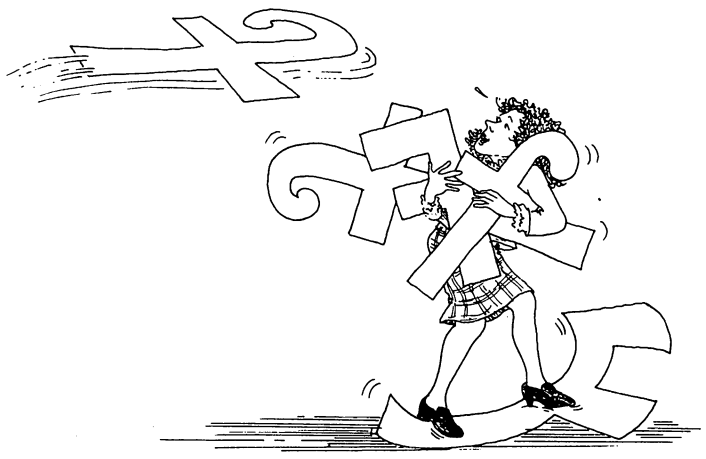
This further generalization can be helpful in the initial design stages of program development. Once the category for a particular design task is identified, it becomes a matter of selecting the appropriate statements and defining their sequence of execution.

5.4 Posers and Problems

1. Assume a program is to be designed that gives credit for a correct answer *only* if it is answered correctly on the first attempt. Outline the statements needed to accomplish this.
2. Assume a program is to be designed that gives a hint on the first miss and the correct answer on the second miss. Outline the statements needed to accomplish this.
3. Assume five questions are to be randomly selected from a one-dimensional array containing ten questions without repetition of any question during a given run. Outline the statements needed to accomplish this. (*Hint:* See Problems 4 and 5 in Chapter 4.)
4. Assume a program is to be designed that:
   a. Stores 15 questions, 15 answers, 15 hints, 3 responses for correct answers, and 3 responses for incorrect answers in one-dimensional arrays.
   b. Will ask a total of 8 questions.
   c. Will randomly select each question.
   d. Will not repeat any question.
   e. Will give a random response if correct.

f. Will give a random response for an incorrect answer and an appropriate hint on the first miss.
g. Will give the correct answer on the second miss.
h. Will give the number of correct answers at the conclusion of the interaction.

Outline the statements needed to accomplish each of the above steps, then combine them into a program and test its execution.



Show and Tell: Problem-Solving and Drill Examples

"The opportunities of man are limited only by his imagination. But so few have imagination that there are ten thousand fiddlers to one composer."
Charles F. Kettering

"The most beaten paths are certainly the surest; but do not hope to scare up much game on them."
Andre Gide

"All that glitters is not gold."
Shakespeare

Think About This (for Fun)
Read this sentence slowly: "Finished files are the result of years of scientific study combined with the experience of years." Now, once and only once, count out loud the F's (and f's) in that sentence. How many are there?

Think About This (Seriously)
Should at least one course in "Computer Literacy" be required for teacher certification in any area?

6.1 Objectives

For the successful completion of this chapter, you should be able to:

1. Describe the purpose or application of instructional computing programs that are
   a. problem-solvers and
   b. drill-and-practice
   (Sections 6.3–6.4).
2. Modify certain programs in this chapter to present information of your choosing.
3. Describe in outline form the sequence of execution for each of the example programs or program fragments in this chapter.

6.2 Some Example Programs and Programming Strategies

One of the important factors determining the success or failure of a given human endeavor is the amount of imagination (originality, creativity, innovation, etc.) that goes into it. This applies not only to education in general and the instructional process in particular, but also to the use of computers in instruction (in general) and the successful design and development of instructional computing programs (in particular).

In Chapters 10 and 11, specific steps will be discussed in which imagination will have an opportunity to spring forth. Before these steps are discussed, however, examine a few sample programs and program fragments that give an introduction to strategies and techniques for five methods in which instructional computing may be applied.

As these programs are examined, please keep in mind the quotations at the beginning of this chapter. These example programs *are* limited—by imagination and space. They are not meant to be the "well-beaten path" for those who choose to follow one. Also, they are certainly not meant to reflect the "gold" of instructional computing applications. However, they might plant an "imaginative seed" to allow one to reach heights of greater glory and reward in developing instructional computing programs.

Many of the examples are intentionally trivial because their content is *not* the point to be made. Rather, the programs illustrate some of the *strategies* that may be used in designing instructional computing programs. The content is left to the individual author who might use or expand upon these strategies.

Although some of the examples may be related to a specific discipline, they should be considered only as "illustrative vehicles." In many cases, the pedagogical strategy used in a program could be applied in general, even though the content may be specific.

6.2.1 A Note About REM Statements

REM statements are very important for documenting a program listing. If carefully used in the program, they allow the reader to follow the sequence of program "events" with greater ease; their use makes the program more readable to the eye. For this reason, REM statements are extensively used in the following example programs. Hopefully, they will allow the user (particularly, the beginning user) to better visualize a program's design and strategy. This extensive use of REM statements may give the impression that the example programs are overly long and complicated. This is *not* the case. More than 50 percent of the statements for most of the programs are REM statements! If the REM statements were removed, most of the example programs would be less than 50 statements. Remember, then, that the REM statements are there to help explain the program as the listing is examined.

It should also be mentioned that REM statements, just like any other statements, require space in the system's memory. If the memory available in a system is limited, and the program design is lengthy, *judicious* use of REM should be made.

Finally, and perhaps foremost, keep in mind that some of these example programs may be modified and expanded for actual class use. The REM statements in the listings will help explain how to do this.

6.3 Problem-Solving Applications

The heaviest use of instructional computing to date is that of problem solving—writing computer programs to solve specific discipline-oriented problems. This particular application, for all practical purposes, has no limits. It could be finding the roots of a quadratic equation in mathematics, calculating lunar orbits in physics, solving gas-law problems in chemistry, analyzing voting behavior in sociology, determining circulation trends in library science, and so on.

6.3.1 PROGRAM 9: Compound Interest

Most, if not all, problem-solving programs are based upon some formula or mathematical expression. Known parameters (elements) of the expression are input or read from data, and the solution to an unknown parameter is calculated and output. Program 9 illustrates a business-oriented problem related to the return on invested capital. The known parameters are:

1. A given principal amount to be invested.
2. A given interest rate.
3. A given number of compounding periods per year.
4. A given number of years.

The future value is calculated and displayed, based on the following formula:

```
F = A * (1 + (R/C)) ↑ (C*Y)
```

where F = future value of the investment
A = the amount to be invested
R = the rate of interest (decimal)
C = number of times the interest rate is compounded annually
Y = number of years the principal is invested

Remember, in the above formula and in statement 210 of the program, the "up arrow" (↑) is the system's way to "raise to the power of."

Also note in the listing of the program:

1. Statement 370 and the method used for printing a "divider" between each calculation.
2. Statements 230–330 (in particular, statement 300), and the method used for "rounding" a value to two decimal places.

RUN from disk and refer to the listing and run of Program 9.

PROGRAM 9

```
10 REM   PROGRAM 9 DESCRIPTION
20 REM  ===============
30 REM  PROBLEM SOLVING: THIS PROGRAM
40 REM  CALCULATES FUTURE VALUES OF
50 REM  INVESTMENTS;  DEMOS 'RVS ON'
60 REM  AND 'RVS OFF' USE VIA CHR$()
70 REM  ===============
80 PRINT "♡"
80 PRINT "    INVESTMENT FUTURE VALUES"
100 PRINT
```

```
110 PRINT "WHAT IS THE AMOUNT INVESTED";
120 INPUT A
130 PRINT "AT WHAT INTEREST RATE (%)";
140 INPUT R
150 PRINT "TIMES COMPOUNDED PER YEAR";
160 INPUT C
170 R = R/100
180 PRINT "FOR HOW MANY YEARS";
190 INPUT Y
200 REM  ===FORMULA FOR CALC===
210 F = A * (1 + (R / C)) ↑ (C * Y)
220 PRINT : PRINT "ITS FUTURE VALUE WOULD BE";
230 REM  ===SET RVS ON DISPLAY===
240 PRINT CHR$(18);
250 REM  ================
260 REM  MULTIPLY THE VALUE OF F BY 100; THEN
270 REM  GET THIS INTEGER VALUE; THEN DIVIDE
280 REM  BY 100.  THIS GIVES A VALUE TO 2 DECIMAL PLACES.
290 REM  ================
300 PRINT "$"; INT (F * 100) / 100;
310 REM  ===SET BACK TO NORMAL DISPLAY===
320 PRINT CHR$(146)
330 PRINT "GIVING YOU A GAIN OF $" INT((F-A) * 100)/100
340 PRINT : PRINT "CALCULATE ANOTHER (Y OR NO)";
350 INPUT A$
360 IF A$<> "Y" THEN 380
370 PRINT : FOR I = 1 TO 40 : PRINT "*"; : NEXT I : GOTO 100
380 END

RUN
   INVESTMENT FUTURE VALUES

WHAT IS THE AMOUNT INVESTED?10000
AT WHAT INTEREST RATE (%)?12
TIMES COMPOUNDED PER YEAR?4
FOR HOW MANY YEARS?10

ITS FUTURE VALUE WOULD BE $32620.37
GIVING YOU A GAIN OF $22620.37

CALCULATE ANOTHER (Y OR N)?Y

****************************************
WHAT IS THE AMOUNT INVESTED?10000
AT WHAT INTEREST RATE (%)?12
TIMES COMPOUNDED PER YEAR?365
FOR HOW MANY YEARS?10
```

```
ITS FUTURE VALUE WOULD BE $33194.63
GIVING YOU A GAIN OF $23194.63

CALCULATE ANOTHER (Y OR N)?N
```

6.3.2 PROGRAM 10: Statistics

The formula for the mean of a set of scores is:

$$\text{Mean} = \frac{\text{Sum of scores}}{\text{Number of scores}}$$

The variance may be found from:

$$\text{Variance} = \frac{\text{Sum of the squared difference between the mean and score}}{\text{Number of scores}}$$

The standard deviation of a set of scores is:

Standard deviation = square root of the variance

The Z-score may be found from:

$$Z = \frac{\text{Difference of score from mean}}{\text{Standard Deviation}}$$

A program may be written to solve for these unknowns, given a set of scores. (Beginning at statements 430–436 and ending at statements 712–716, note how the output from this program may be directed to a printer. Note the method used to compute values to a number of decimal places in statements 580–590 and 700.)

RUN from disk and refer to the listing and run of Program 10.

PROGRAM 10

```
10 REM   PROGRAM 10
20 REM  ================
30 REM  PROBLEM SOLVING: THIS PROGRAM
40 REM  CALCULATES MEAN, VARIANCE
50 REM  STANDARD DEVIATION, AND
60 REM  Z-SCORES FOR A SET OF
70 REM  SCORES.
80 REM  ================
90 REM
100 REM  ================
110 REM  VARIABLE DICTIONARY
120 REM  ================
130 REM  D    - STANDARD DEVIATION
140 REM         (SQ. ROOT OF VARIANCE)
150 REM  D1() - DISTANCE OF A GIVEN
160 REM         SCORE FROM THE MEAN
```

```
170 REM  M    - MEAN OF SCORES
180 REM  S()  - SCORES ENTERED VIA INPUT
190 REM  T    - CUMULATIVE TOTAL OF SCORES
200 REM  T1   - CUMULATIVE TOTAL OF THE SQUARE
210 REM         OF THE DISTANCE OF A GIVEN
220 REM         SCORE FROM THE MEAN
230 REM  V    - VARIANCE
240 REM  ===============
250 REM
260 DIM S(100),D1(100)
270 PRINT "♡";
280 PRINT "MEAN, VARIANCE, AND STANDARD"
290 PRINT "DEVIATION OF A SET OF SCORES"
300 PRINT
310 PRINT "ENTER THE SCORES, TO STOP,"
320 PRINT "  ENTER ANY NEGATIVE NO."
330 REM  ===ROOM FOR 100 SCORES===
340 FOR I = 1 TO 100
350 PRINT "SCORE (NEGATIVE TO STOP)";
360 INPUT S(I)
370 IF S(I) < 0 THEN 420
380 REM  ===CUMULATIVE SCORE TOTAL===
390 T = T + S(I)
400 NEXT I
410 REM  ===THE VALUE OF I MUST BE DECREASED BY 1===
420 I = I - 1
430 PRINT "♡" : INPUT "WANT A PRINTER COPY (Y OR N)";Z$
432 IF Z$ <>"Y"  THEN 440
434 OPEN 4,4
436 CMD4
440 PRINT "MEAN","VAR","SD"
450 PRINT "----","---","--"
460 REM  ===COMPUTE THE MEAN===
470 M = T / I
480 FOR J = 1 TO I
490 REM  ===GET DISTANCE OF EACH SCORE FROM MEAN===
500 D1(J) = S(J) - M
510 REM  ===BUILD A CUMULATIVE TOTAL===
520 T1 = T1 + D1(J)↑ 2
530 NEXT J
540 REM  ===NEXT COMPUTE VARIANCE===
550 V = T1 / I
560 REM  ===COMPUTE STANDARD DEVIATION===
570 D = SQR (V)
580 REM  ===SET THE VALUES TO 2 DECIMAL PLACES===
590 PRINT  INT (M * 100) / 100, INT (V * 100) / 100, INT (D
    * 100) / 100
600 PRINT
610 PRINT "     Z-SCORES"
```

```
620 PRINT "SCORE","Z-SCORE"
630 PRINT "-----","-------"
640 FOR J = 1 TO I
650 REM  ===============
660 REM  Z-SCORE IS THE DISTANCE OF A
670 REM  GIVEN SCORE FROM THE MEAN,
680 REM  DIVIDED BY THE STANDARD DEVIATION.
690 REM  ===============
700 PRINT S(J), INT (D1(J) / D * 100) / 100
710 NEXT J
712 IF Z$ <> "Y" THEN 720
714 PRINT#4
716 CLOSE4
720 PRINT : INPUT "ANALYZE ANOTHER SET (Y OR N)";Z$
730 IF Z$ = "Y" THEN T = 0 : T1 = 0 : GOTO 300
740 END
```

```
RUN
MEAN, VARIANCE AND STANDARD
DEVIATION OF A SET OF SCORES

ENTER THE SCORES. TO STOP,
  ENTER ANY NEGATIVE NO.
SCORE (NEGATIVE TO STOP)?100
SCORE (NEGATIVE TO STOP)?90
SCORE (NEGATIVE TO STOP)?80
SCORE (NEGATIVE TO STOP)?70
SCORE (NEGATIVE TO STOP)?60
SCORE (NEGATIVE TO STOP)?-1

MEAN           VAR            SD
----           ---            --
80             200            14.14

     Z-SCORES
SCORE          Z-SCORE
-----          -------
100             1.41
90              .7
80             0
70              -.71
60              -1.42

ANALYZE ANOTHER SET (Y OR N)?Y

ENTER THE SCORES. TO STOP,
  ENTER ANY NEGATIVE NO.
SCORE (NEGATIVE TO STOP)?98
SCORE (NEGATIVE TO STOP)?78
SCORE (NEGATIVE TO STOP)?67
SCORE (NEGATIVE TO STOP)?55
SCORE (NEGATIVE TO STOP)?99
SCORE (NEGATIVE TO STOP)?100
SCORE (NEGATIVE TO STOP)?88
SCORE (NEGATIVE TO STOP)?90
SCORE (NEGATIVE TO STOP)?-1

MEAN            VAR             SD
----            ---             --
84.37           236.73          15.38

     Z-SCORES
SCORE           Z-SCORE
-----           -------
98               .88
78               -.42
87               -1.13
55               -1.91
99               .95
100              1.01
88               .23
90               .36

ANALYZE ANOTHER SET (Y OR N)?N
                BYE-BYE
```

6.3.3 PROGRAM 11: File Maintenance

One problem that teachers face is ease of access to, and updating of, student records. Program 11 is one approach to letting a computer program do most of the work (the grades still have to be entered somehow—via a keyboard in this case). This program is included here to serve more as a utility program that the reader can use than as an illustrative example, because some of the statements are more advanced than those of Chapters 1–5. These additional statements are:

1. OPEN
2. INPUT#
3. PRINT#
4. CLOSE

These new statements are needed to access another file, called TESTS, in which the record information is kept. The concept of this file is the same as that of DATA-READ program statements. However, the data (student records) are not accessed from DATA statements in the program but from the file TESTS. Use of such files allows data to be updated for use in a program without having to rewrite or add DATA statements in the body of the program.

Refer to the REM statements in Program 11; we will briefly explain here how these new statements are used.

1. The OPEN statement defines whether information is to be input (R) *from* the file or output (W) *to* the file, the number (n) to be assigned to the file (for our purposes, this will usually be 2), and the file name. (See statements 537–570.)
2. The INPUT# n, <variable name> instructs the program to *retrieve* (read *from* the file) the next element of information and assign it to the defined variable. (See statements 670–740.)
3. The PRINT# n, <variable name> instructs the program to *place* into the file (write *to* the file) the current value assigned to the defined variable. (See statements 1540–1590.)
4. The CLOSE statement simply "closes" the file, indicating that information is no longer needed to be read from or written to it. (See statements 750 and 1600.)

Note! The INPUT# and PRINT# statements refer to *file* "reading" and "writing." Do *not* confuse them with the more familiar INPUT and PRINT statements; they are *not* the same.

The file TESTS contains numerical data in the following sequence: n1,s1,s2, . . .,n2,s1,s2, . . . , etc. Here, n1 is the number of scores for the first student name, s1,s2, . . . are the scores for that student, n2 is the number of scores for the second student name and s1,s2, . . . are the scores for that student, and so on. An example might look like: 2,99,88,1,75,4,87,85,92,95 . . . etc.

Program 11 appears to be lengthy, but, again, this is due to extensive use of REM statements to help explain the program's execution. The program may be easily modified for actual class use by changing the DIM and DATA statements to meet the user's needs and running the program INITIALZ to erase the sample data in the file TESTS.

RUN from disk and refer to the listing and run of Program 11.

PROGRAM 11

```
10 REM  PROGRAM 11
20 REM  ==============
30 REM  PROBLEM SOLVING:
40 REM  THIS PROGRAM IS AN EXAMPLE OF RECORD
50 REM  KEEPING USING SEQUENTIAL FILES.
60 REM  FIVE STUDENT NAMES ARE DATA ELEMENTS
70 REM  IN THE BODY OF THE PROGRAM.  THE
80 REM  NUMBER OF SCORES FOR EACH STUDENT AND
90 REM  THEIR RESPECTIVE SCORES ARE STORED
100 REM  SEQUENTIALLY IN THE FILE "TESTS"
```

```
110 REM  ON THE DISK. THE PROGRAM MAY BE USED
120 REM  FOR REAL STUDENT RECORD KEEPING BY
130 REM  CHANGING THE DIM AND DATA STATEMENTS
140 REM  ACCORDINGLY, AND THEN RUNNING THE PROGRAM
150 REM  "INITIALZ"
160 REM  ===============
170 REM  VARIABLE DICTIONARY
180 REM  ===============
180 REM  N$     - STUDENT NAME SOUGHT (VIA INPUT)
200 REM  N$     - STUDENT NAMES FROM PROGRAM DATA
210 REM           STATEMENTS
220 REM  N()    - NUMBER OF SCORES FOR EACH
230 REM           STUDENT (FROM FILE "TESTS")
240 REM  P      - NUMBER OF STUDENTS (FROM PROGRAM
250 REM           DATA STATEMENT)
260 REM  S(,)   - TWO-DIM ARRAY: ROW IS STUDENT
270 REM           NUMBER; COLUMN IS NUMBER OF
280 REM           SCORES FOR THAT STUDENT
290 REM  ==============
300 REM  PROGRAM EXAMPLE DIMENSIONS FOR A
310 REM  MAXIMUM OF 20 STUDENTS AND 8 SCORES.
320 REM  CHANGE DIM IF USED IN REAL CLASS
330 REM  ===============
340 PRINT "♡": DIM S(20,8),N$(20),N(20)
350 PRINT "C L A S S  S C O R E  K E E P I N G"
360 REM  ===============
370 REM  DATA FOR NUMBER OF STUDENTS.  CHANGE
380 REM  IF USED IN REAL CLASS.
380 REM  ===============
400 DATA 5
410 REM  ===============
420 REM  READ THE NUMBER OF STUDENTS;THEN
430 REM  STORE THE NAMES IN N$( ).
440 REM  ===============
450 READ P
460 REM  ===============
470 REM  5 STUDENT NAMES (CHANGE FOR REAL USE!)
480 DATA "CANTOR","DARWIN","EDGAR","MCCARTHY","ZILLA"
490 REM  ===============
500 FOR I=1 TO P
510 READ N$(I)
520 NEXT I
530 REM  ===============
540 REM  FIRST OPEN THE FILE "TESTS" WHICH
550 REM  CONTAINS STUDENT SCORE DATA
560 REM  ===============
570 OPEN 2,8,2,"0:TESTS,S,R"
580 REM  ===============
590 REM  THEN ISSUE THE "COMMANDS" TO
```

```
600 REM  START READING THE DATA
610 REM  FROM IT.  DATA ARE STORED IN THE
620 REM  SEQUENCE: (NUMBER OF SCORES FOR
630 REM  A STUDENT), (EACH SCORE FOR
640 REM  THAT STUDENT).  N(I) IS THE NUMBER OF
650 REM  SCORES;  S(I,J) IS THE SCORE.
660 REM  ===============
670 FOR I=1 TO P
680 REM  ===GET THE NUMBER OF SCORES FROM FILE===
690 INPUT#2, N(I)
700 REM  ===NOW GET EACH SCORE FROM FILE===
710 FOR J=1 TO N(I)
720 INPUT#2, S(I,J)
730 NEXT J
740 NEXT I
750 CLOSE 2
780 REM  ===============
790 REM  FINISHED READING DATA
800 REM  ================
810 PRINT
820 PRINT "DO YOU WANT TO:"
830 PRINT "1. ENTER NEW SCORES"
840 PRINT "2. RETRIEVE SCORES"
850 PRINT "3. STOP"
855 PRINT:PRINT "(ENTER 1-3)";
860 INPUT C
870 IF C=1 THEN 910
880 IF C=2 THEN 910
890 IF C=3 THEN 1250
900 GOTO 810
910 PRINT
920 PRINT "STUDENT'S NAME (OR STOP)";
930 INPUT N$
940 IF N$ = "STOP" THEN 810
850 FOR I=1 TO P
960 REM  ===MATCH FOUND WITH NAMES?===
970 IF N$ = N$(I) THEN 1010
980 NEXT I
990 PRINT N$ " IS NOT ON FILE!"
1000 GOTO 920
1010 IF C=1 THEN 1190
1020 REM  ================
1030 REM  PRINT THE SCORES FOR THE STUDENT
1040 REM  ===============
1050 PRINT "SCORES FOR " N$(I) ":"
1055 IF N(I) = 0 THEN PRINT "*** NO SCORES RECORDED ***" :
     GOTO 910
1060 FOR J=1 TO N(I)
1070 PRINT S(I,J);"♡";
```

```
1080 REM  ===CUMULATIVE TOTAL FOR STUDENT===
1090 T = T + S(I,J)
1100 NEXT J
1130 PRINT "  AVE = ";T/N(I)
1140 T=0
1141 PRINT : INPUT "NEED TO EDIT THESE SCORES (Y/N)";Z$
1142 IF Z$ <>"Y" THEN 910
1143 FOR K = 1 TO N(I)
1144 PRINT "EDIT THIS SCORE: " S(I,K) "(Y/N)"; : INPUT Z$
1145 IF Z$ <> "Y" THEN 1147
1146 INPUT "SCORE SHOULD BE WHAT VALUE";S(I,K)
1147 NEXT K
1150 GOTO 1050
1160 REM  ================
1170 REM  ADD MORE SCORES FOR STUDENTS
1180 REM  ================
1190 PRINT "NEXT SCORE FOR ";N$;
1200 REM  ===INCREASE THE SCORE COUNT BY 1===
1210 N(I)=N(I)+1
1220 REM  ===STORE NEW SCORE IN ARRAY===
1230 INPUT S(I,N(I))
1240 GOTO 910
1250 PRINT
1252 PRINT
1254 PRINT "USE PRINTER (Y OR N)";
1260 INPUT Z$
1262 IF Z$<>"Y" THEN 1270
1264 OPEN 4,4
1266 CMD4
1270 REM  ================
1280 REM  PRINT OUT CLASS RECORDS
1290 REM  ================
1300 FOR I = 1 TO P
1310 FOR K = 1 TO 20 : PRINT "/"; : NEXT K : PRINT
1320 PRINT "NAME: " N$(I) : PRINT "SCORES:";
1325 IF N(I) = 0 THEN PRINT "*** NO SCORES RECORDED ***" :
     GOTO 1450
1330 FOR J=1 TO N(I)
1340 PRINT S (I,J) "♡";
1350 REM ===TOTALS FOR EACH STUDENT AND COMPLETE CLASS===
1360 T = T + S(I,J) : T1 = T1 + S(I,J)
1370 NEXT J
1410 PRINT : PRINT "AVERAGE = " T/N(I)
1420 REM  ===CUMULATIVE TOTAL SCORE NUMBER===
1430 S1 = S1 + N(I)
1440 T = 0
1450 NEXT I
1460 PRINT
1470 PRINT "THE CLASS AVERAGE IS ";INT((T1/S1) * 100)/100
```

```
1472 IF Z$<>"Y" THEN 1480
1474 PRINT#4
1476 CLOSE 4
1480 REM  ===============
1490 REM  WRITE ALL DATA WITH UPDATES
1500 REM  BACK TO THE FILE "TESTS".
1510 REM  ===============
1520 OPEN 2,8,2,"@0:TESTS,S,W"
1540 FOR I=1 TO P
1550 PRINT#2,N(I)
1560 FOR J=1 TO N(I)
1570 PRINT#2,S(I,J)
1580 NEXT J
1590 NEXT I
1600 CLOSE 2
1610 PRINT:PRINT "     *** D O N E ***"
1620 END
```

```
RUN
C L A S S  S C O R E  K E E P I N G
DO YOU WANT TO:
1. ENTER NEW SCORES
2. RETRIEVE SCORES
3. STOP (ENTER 1-3)?1

STUDENT'S NAME (OR STOP)?BERGEN
BERGEN IS NOT ON FILE!
STUDENT'S NAME (OR STOP)?DARWIN
NEXT SCORE FOR DARWIN?92

STUDENT'S NAME (OR STOP)?STOP
DO YOU WANT TO:
1. ENTER NEW SCORES
2. RETRIEVE SCORES
3. STOP (ENTER 1-3)?2

STUDENT'S NAME (OR STOP)?DARWIN
SCORES FOR DARWIN:
92   AVE = 92

NEED TO EDIT THESE SCORES (Y/N)? Y
EDIT THIS SCORE: 92 (Y/N)?Y
SCORE SHOULD BE WHAT VALUE?94
SCORES FOR DARWIN:
94   AVE = 94

NEED TO EDIT THESE SCORES (Y/N)? N
```

```
STUDENT'S NAME (OR STOP)?ZILLA
SCORES FOR ZILLA:
*** NO SCORES RECORDED ***

STUDENT'S NAME (OR STOP)?STOP
DO YOU WANT TO:
1. ENTER NEW SCORES
2. RETRIEVE SCORES
3. STOP (ENTER 1-3)?1

STUDENT'S NAME (OR STOP)?ZILLA
NEXT SCORE FOR ZILLA?100

STUDENT'S NAME (OR STOP)?ZILLA
NEXT SCORE FOR ZILLA?90

STUDENT'S NAME (OR STOP)?EDGAR
NEXT SCORE FOR EDGAR?80

STUDENT'S NAME (OR STOP)?MCCARTHY
NEXT SCORE FOR MCCARTHY?70

STUDENT'S NAME (OR STOP)?STOP
DO YOU WANT TO:
1. ENTER NEW SCORES
2. RETRIEVE SCORES
3. STOP (ENTER 1-3)?3

USE PRINTER (Y OR N)?N

////////////////////
NAME: CANTOR
SCORES: *** NO SCORES RECORDED ***
////////////////////
NAME: DARWIN
SCORES: 94
AVERAGE = 94
////////////////////
NAME: EDGAR
SCORES: 80
AVERAGE = 80
////////////////////
NAME; MCCARTHY
SCORES: 70
AVERAGE = 70
////////////////////
NAME: ZILLA
SCORES: 100 90
AVERAGE = 95
```

```
THE CLASS AVERAGE IS 86.8
     *** D O N E ***
```

6.4 Drill-and-Practice Applications

Drill-and-practice programs are second only in use to problem-solving applications in instructional computing. This technique also has wide application in any area in which certain fundamental concepts require practice for mastery, such as multiplication tables, chemical nomenclature, Latin-English word root translations, state capitals, and so on.

Drill-and-practice programs are generally very straightforward. An introduction, usually including examples, is given; drill questions are presented (either linearly or by random selection); answers are entered and checked for accuracy; appropriate feedback is given; the next question is asked; and, at the end of the program, some form of performance report is given and, perhaps, recorded.

6.4.1 PROGRAM 12: Linear Selection of Drill Questions

Drill programs can be easily constructed with DATA-READ and FOR-NEXT statements. If the questions are to be linear, the DATA consists of question-answer pairs on any chosen topic that are READ as part of a FOR-NEXT question sequence. A skeleton program might be:

```
 10 DATA ["Question 1","Answer 1","Question 2", etc.]
  .
  .
  .
200 PRINT "[Introductory statements, examples, etc.]"
  .
  .
  .
500 FOR I = 1 TO [Number of questions to ask]
510    READ Q$,A$
520    PRINT Q$;
530    INPUT R$
540    IF R$ = A$ THEN PRINT "EXCELLENT!" : C = C + 1
          : GOTO 550
544    PRINT "A CORRECT ANSWER IS ";A$
550 NEXT I
560 PRINT "YOU ANSWERED ";C;" QUESTIONS CORRECTLY!"
570 END
```

In the following program on state capitals (Program 12), note the use of the RND(0) function to "flip a coin" to determine if the state or the capital is to be asked as a question (see statements 330–350). In the normal READ sequence (as defined in this program), Q$ (the question) contains the state, and A$ (the answer) contains the capital. If the question-answer values are to be reversed, the contents of the variables Q$ and A$ must be switched. This switch is accomplished in statements 380–460 by the use of a dummy variable, D$, to hold the original question (value of Q$) as the switching process is done.

Again, please remember that by just changing the contents of the DATA statements, it is possible to make the program more than just a trivial drill on state capitals!

RUN from disk and refer to the listing and run of Program 12.

PROGRAM 12

```
10 REM  PROGRAM 12
20 REM  ===============
30 REM  DRILL AND PRACTICE:
40 REM  PROGRAM DEMOS THE LINEAR QUESTION-
50 REM  AND-ANSWER SEQUENCE VIA DATA-READ,
60 REM  PLUS SWITCHING A QUESTION FOR AN
70 REM  ANSWER AND AN ANSWER FOR A QUESTION.
80 REM
90 REM  ===============
100 REM  VARIABLE DICTIONARY
110 REM  ===============
120 REM  A$ - ANSWER  (FROM DATA-READ)
130 REM  C  - NUMBER CORRECT COUNTER
140 REM  D$ - HOLDS ORIGINAL QUESTION IN
150 REM       QUESTION/ANSWER SWITCHING
160 REM  Q$ - QUESTION (FROM DATA-READ)
170 REM  R$ - USER RESPONSE (VIA INPUT)
180 REM  ===============
190 DATA "TEXAS","AUSTIN","ARKANSAS","LITTLE ROCK"
200 DATA "NEW MEXICO","SANTA FE","OKLAHOMA"
210 DATA "OKLAHOMA CITY","OREGON","SALEM"
220 PRINT "♡"
230 REM  ===INTRODUCTION===
240 PRINT "STATE CAPITAL DRILL"
250 PRINT
260 PRINT "IF I GIVE THE STATE, YOU GIVE"
270 PRINT "THE CAPITAL; IF I GIVE THE"
280 PRINT "CAPITAL, YOU GIVE THE STATE."
290 PRINT
300 FOR I = 1 TO 5
310 READ Q$,A$
320 PRINT
```

```
330 REM  ==='FLIP' A COIN===
340 X = INT(2 * RND(0) + 1)
350 IF X = 2 THEN 490
360 REM  ===DO THE SWITCH IF X IS 1===
370 REM  ===============
380 REM  HERE'S THE SWITCH...STORE Q$ IN
400 REM  D$, THEN ASSIGN A$ TO Q$ (ANSWER
410 REM  IS NOW QUESTION). NEXT ASSIGN D$ TO A$
420 REM  (NOW THE ORIGINAL QUESTION IS THE ANSWER).
430 REM  ===============
440 D$ = Q$
450 Q$ = A$
460 A$ = D$
470 REM  ===SWITCH COMPLETED===
480 REM  ===NOW ASK THE QUESTION===
480 PRINT Q$;
500 INPUT R$
510 IF R$ = A$ THEN 540
520 PRINT "A CORRECT ANSWER IS ";A$
530 GOTO 560
540 PRINT "GREAT!"
550 C=C+1
560 NEXT I
570 PRINT "YOU GOT ";C;" CORRECT!"
580 END

RUN
STATE CAPITAL DRILL

IF I GIVE THE STATE, YOU GIVE
THE CAPITAL; IF I GIVE THE
CAPITAL, YOU GIVE THE STATE.

TEXAS?AUSIN
A CORRECT ANSWER IS AUSTIN

ARKANSAS?LITTLE ROCK
GREAT!

NEW MEXICO?SANTA FE
GREAT!

OKLAHOMA?NORMAN
A CORRECT ANSWER IS OKLAHOMA CITY

OREGON?SALEM
GREAT!
YOU GOT 3 CORRECT!
```

6.4.2 PROGRAM 13: Random Selection of Drill Questions

If the questions are to be randomly selected from a bank of data elements, the DATA are READ into one-dimensional arrays prior to presentation of the question sequence. In simplest form, a skeleton program could be:

```
 10 DIM Q$[Number of questions in bank],A$( ),Z( )
 20 DATA ["Question 1","Answer 1","Question 2", etc.]
  .
  .
  .
150 FOR I = 1 TO [Number of questions in bank]
170    READ Q$(I),A$(I)
180    Z(I) = 0
190 NEXT I
200 PRINT "[Introductory statements, examples, etc.]"
  .
  .
  .
500 FOR I = 1 TO [No. of questions to be asked]
505 REM   RANDOMLY SELECT A QUESTION NUMBER
510    J = INT([No. of questions in bank]*RND(0) + 1)
515 REM   HAS THIS NUMBER ALREADY BEEN SELECTED?
520    IF Z(J) = 1 THEN 510
530    Z(J) = 1
540    PRINT Q$(J);
550    INPUT R$
560    IF R$ = A$(J) THEN PRINT "GREAT!" : C = C + 1
         : GOTO 570
564    PRINT "A CORRECT ANSWER IS ";A$(J)
570 NEXT I
580 PRINT "YOU ANSWERED ";C;" QUESTIONS CORRECTLY!"
590 END
```

Note in the example, Program 13, that only three (see statement 350) of the five possible questions are randomly selected. In general, you should have approximately 25 percent more questions in the bank than are to be asked by random selection. This practice will reduce the time required for the program to find a question that has not been asked previously.

RUN from disk and refer to the listing and run of Program 13.

PROGRAM 13

```
10 REM  PROGRAM 13
20 REM  ===============
30 REM  DRILL AND PRACTICE:
40 REM  PROGRAM DEMOS RANDOM SELECTION OF
50 REM  QUESTIONS/ANSWERS FROM ONE-DIM ARRAYS
```

```
60 REM  WITHOUT REPEATING ANY QUESTION.
70 REM  SWITCHING OF QUES/ANSWER OCCURS AT RANDOM.
80 REM  ===============
90 REM  VARIABLE DICTIONARY
100 REM  ===============
110 REM  A$(J) - ANSWER TO QUESTION
120 REM  D$    - HOLDS QUESTION TEMPORARILY
130 REM          IN QUES/ANSWER SWITCHING
140 REM  J     - RANDOM INTEGER VALUE
150 REM  Q$(J) - RANDOM QUESTION FROM LIST
160 REM  Z(J)  - FLAG FOR SELECTED INTEGER
170 REM  ==============
180 DIM Q$(5),A$(5),Z(5)
190 REM  ===INTRODUCTION SECTION===
200 PRINT "♡"
210 PRINT "STATE CAPITAL DRILL"
220 PRINT
260 PRINT "IF I GIVE THE STATE, YOU GIVE"
270 PRINT "THE CAPITAL; IF I GIVE THE"
280 PRINT "CAPITAL, YOU GIVE THE STATE."
290 PRINT
300 REM   ===STORE THE QUESTIONS/ANSWERS===
310 FOR I=1 TO 5
320 READ Q$(I),A$(I)
330 NEXT I
340 REM  ===ASK ONLY 3 OF THE POSSIBLE 5===
350 FOR I=1 TO 3
380 PRINT
370 REM  ===SELECT A QUESTION NUMBER===
380 J=INT(5*RND(0)+1)
390 REM  ===HAS IT BEEN SELECTED BEFORE?===
400 IF Z(J)=1 THEN 380
410 REM  ===FLAG J AS A SELECTED NUMBER===
420 Z(J)=1
440 REM  ==='FLIP' A COIN===
450 X=INT(2*RND(0)+1)
480 IF X=2 THEN 520
470 REM  ===DO THE SWITCH IF X IS 1===
480 D$=Q$(J)
490 Q$(J)=A$(J)
500 A$(J)=D$
510 REM  ===ASK THE QUESTION===
520 PRINT Q$(J);
530 INPUT R$
540 IF R$=A$(J) THEN 570
550 PRINT "A CORRECT ANSWER IS ";A$(J)
560 GOTO 590
570 PRINT "GREAT!"
580 C=C+1
```

```
590 NEXT I
600 PRINT "YOU GOT ";C;" CORRECT!"
610 END
620 DATA "TEXAS","AUSTIN","ARKANSAS","LITTLE ROCK"
630 DATA "NEW MEXICO,",SANTA FE","OKLAHOMA"
640 DATA "OKLAHOMA CITY","OREGON","SALEM"

RUN
STATE CAPITAL DRILL

IF I GIVE THE STATE, YOU GIVE
THE CAPITAL; IF I GIVE THE
CAPITAL, YOU GIVE THE STATE.

TEXAS?AUSTIN
GREAT!

OREGON?SALEM
GREAT!

SANTA FE?NEW JERSEY
A CORRECT ANSWER IS NEW MEXICO
YOU GOT 2 CORRECT!

RUN
STATE CAPITAL DRILL

IF I GIVE THE STATE, YOU GIVE
THE CAPITAL; IF I GIVE THE
CAPITAL, YOU GIVE THE STATE.

NEW MEXICO?SANTA FE
GREAT!

LITTLE ROCK?ARKANSAS
GREAT!

SALEM?OREGON
GREAT!
YOU GOT 3 CORRECT!
```

6.4.3 PROGRAM 14: User Options and Subroutines

Use of GOSUB-RETURN routines also makes development of *linear* drill-and-practice programs a simple task. The minimum needed could be:

1. PRINT statements for the introduction, examples, and question content:

```
10 PRINT "[Introductory statements]"
  .
  .
  .
100 PRINT "[Presenting examples]"
  .
  .
  .
200 PRINT "[Ask question 1]"
```

2. Assign a correct answer to a variable:

```
210 A$="[Answer to question 1]"
```

3. Go to the answer checking subroutine:

```
220 GOSUB 10000
```

4. Present the next question following RETURN:

```
230 PRINT "[Ask question 2]"
```

5. Assign answer to a variable:

```
240 A$="[Answer to question 2]"
```

6. Go to the subroutine again:

```
250 GOSUB 10000
  .
  .
  .
etc.
```

The subroutine allows for answer input:

```
10000 INPUT R$
```

Checks for accuracy:

```
10010 IF R$=A$ THEN 10040
```

Presents a correct answer if missed:

```
10020 PRINT "A CORRECT ANSWER IS ";A$
```

Returns to ask the next question:

```
10030 RETURN
```

Gives a positive feedback if correct:

```
10040 PRINT "VERY GOOD!"
```

Increases a number correct counter by one:

```
10050 C = C + 1
```

And returns to ask the next question:

```
10060 RETURN
```

These statements outline a general design sequence. But, to a user, a drill-and-practice program that just asks a question, says "CORRECT" or "INCORRECT," and then asks the next question can be awfully boring. However, we can liven up the program by giving the user some options, such as: SKIP (a question), ANSWER (to a question), and STOP (at will). Of course, we can also randomize the positive feedback and use some method for showing simple clues in asking a question. Examples of these types of additions are shown in Program 14.

Note in the program listing that any user response is *first* checked for a match with a defined "option," then for a correct answer match. Otherwise, a requested option that was input by a user might be considered as only an incorrect answer to the question. (See statements 4010–4030.) Finally, note the use of four different subroutines: one for positioning (1000), one for pausing (2000), one for displaying options (3000), and one for response input and checking (4000).

RUN from disk and refer to the listing and run of Program 14.

PROGRAM 14

```
10 REM  PROGRAM 14
20 REM  ===============
30 REM  DRILL AND PRACTICE:
40 REM  THIS PROGRAM DEMONSTRATES USE OF
50 REM  MULTIPLE SUBROUTINES. USER OPTIONS
60 REM  FOR PROGRAM CONTROL ARE ALSO SHOWN.
70 REM
80 REM  ===============
90 REM  VARIABLE DICTIONARY
100 REM ===============
110 REM A$   - THE CORRECT ANSWER
120 REM C    - COUNTER FOR CORRECT ANSWERS
150 REM P$()- LIST OF POSITIVE REINFORCERS
160 REM R    - A RANDOM NUMBER (4-1)
170 REM R$   - USER'S RESONSE
180 REM S$   - DASHES REPRESENTING CORRECT ANSWER
190 REM ===============
200 REM DIMENSION FEEDBACK LIST; ASSIGN VALUES
210 REM ===============
220 DIM P$(4)
230 DATA "GREAT","WONDERFUL","HOT-DOG","THAT'S IT"
240 FOR I = 1 TO 4 : READ P$(I) : NEXT I
250 REM ===============
260 REM SUBROUTINE 1000 FOR SCREEN POSITIONING
270 REM ===============
280 PRINT "♡" : GOSUB 1000
280 PRINT " A  B I T  O F  H I S T O R Y"
300 REM ===============
310 REM SUBROUTINE 2000 FOR PAUSING
320 REM ===============
330 GOSUB 2000
340 REM ===============
350 REM SUBROUTINE 3000 FOR PRESENTING USER
360 REM OPTIONS AT TOP OF SCREEN
370 REM ===============
380 GOSUB 3000
390 GOSUB 1000
400 REM ===============
410 REM SEQUENCE FOR ALL QUESTIONS IS:
420 REM    1. PRINT(S) TO ASK THE QUESTION
430 REM    2. CORRECT ANSWER ASSIGNED TO A$
440 REM    3. DASHES REPRESENTING ANSWER ASSIGNED TO S$
450 REM    4. GOSUB 4000 FOR INPUT AND ANSWER CHECKING
460 REM ===============
470 REM
480 PRINT "AMERICA'S FIRST PRESIDENT WAS NAMED:"
490 A$ = "GEORGE WASHINGTON"
500 S$ = "------ ----------"
520 REM ===============
530 REM SUBROUTINE 4000 FOR INPUT AND ANSWER CHECKING
540 REM ===============
550 GOSUB 4000
560 GOSUB 3000
570 GOSUB 1000
580 PRINT "AS A YOUNG LAD, HE CHOPPED DOWN A:"
580 A$ = "CHERRY TREE" : S$ = "------ ----"
600 GOSUB 4000
610 GOSUB 3000
620 GOSUB 1000
630 PRINT "HOWEVER, HE WAS CAUGHT RED-HANDED (NO"
640 PRINT "PUN INTENDED) AND COULD NOT TELL A:"
650 A$ = "LIE" ; S$ ="---"
660 GOSUB 4000
670 REM ===ROOM FOR MORE QUESTIONS===
960 PRINT "♡" : GOSUB 1000
970 PRINT,"PERFORMANCE:" : PRINT
990 PRINT "YOU ANSWERED" C "QUESTION(S)!"
990 END
999 REM ===SCREEN POSITIONING SUBROUTINE===
1000 FOR I = 1 TO 8 : PRINT : NEXT I
1010 RETURN
1999 REM ===PAUSING SUBROUTINE===
2000 FOR I = 1 TO 1000 : NEXT I
2010 RETURN
```

```
2999 REM ===OPTION DISPLAY SUBROUTINE===
3000 PRINT "♡"
3010 PRINT "INPUT OPTIONS: 'ANSWER' 'SKIP' 'STOP'"
3020 RETURN
3999 REM ===INPUT AND CHECKING SUBROUTINE===
4000 PRINT TAB(12) S$
4010 PRINT TAB(10) : INPUT R$
4020 IF R$ = "ANSWER" THEN 4070
4030 IF R$ = "SKIP" THEN 4170
4040 IF R$ = "STOP" THEN PRINT "♡" : GOTO 960
4050 IF R$ = A$ THEN 4120
4060 PRINT
4070 PRINT "A CORRECT ANSWER IS " A$ : PRINT
4080 INPUT "PRESS RETURN TO CONTINUE";Z$ : GOTO 4170
4090 REM ===============
4100 REM CORRECT ANSWER PROCEDURE
4110 REM ===============
4120 PRINT "♡" : GOSUB 1000
4130 R = INT(4 * RND(0) + 1)
4140 PRINT TAB(15) P$(R)"!"
4150 C = C + 1
4160 GOSUB 2000
4170 RETURN
```

```
 RUN
 A  B I T  O F   H I S T O R Y
INPUT OPTIONS: 'ANSWER' 'SKIP' 'STOP'

AMERICA'S FIRST PRESIDENT WAS NAMED:
           ------ ----------
          ?GEORGE WASHINGTON

           HOT-DOG!
INPUT OPTIONS: 'ANSWER' 'SKIP' 'STOP'

AS A YOUNG LAD, HE CHOPPED DOWN A:
            ------ ----
           ?CHERRY TREE
           THAT'S IT!
INPUT OPTIONS: 'ANSWER' 'SKIP' 'STOP'

HOWEVER, HE WAS CAUGHT RED-HANDED (NO
PUN INTENDED) AND COULD NOT TELL A:
                ---
               ?FIB

A CORRECT ANSWER IS LIE

PRESS RETURN TO CONTINUE
```

```
                .
                .
                .
        PERFORMANCE:

YOU ANSWERED 2 QUESTION(S)!
```

6.4.4 PROGRAM 15: Random Generation of Question Parameters

For drill programs using numerical values, the RND(0) function may be used to ensure that no two RUNs of the program are identical. That is, although the text of the problem may be the same, the parameters are randomly generated so that each problem appears unique. As an example, consider Program 15, which gives the base and height of a triangle and asks for the area (Area = 1/2 base × height).

Note the method used for showing the correct solution to problems that are missed (see statements 520–560).

RUN from disk and refer to the listing and run of Program 15.

PROGRAM 15

```
10 REM  PROGRAM 15
20 REM  ============
30 REM  DRILL AND PRACTICE
40 REM  PROGRAM DEMOS USER CONTROL OF THE
50 REM  NUMBER OF QUESTIONS TO BE ASKED
60 REM  AND USE OF RND(0) TO RANDOMLY
70 REM  GENERATE NUMBERS WITHIN LIMITS FOR
80 REM  USE IN THE TEXT OF A QUESTION.
90 REM  THE PROGRAM ALSO COMPUTES A PERCENTAGE
100 REM SCORE FOR THE NUMBER OF CORRECT ANSWERS.
110 REM  =============
120 REM  VARIABLE DICTIONARY
130 REM  =============
140 REM  A - AREA OF TRIANGLE
150 REM  B - RANDOMLY SELECTED VALUE FOR
160 REM      THE 'BASE' OF A TRIANGLE
170 REM  H - RANDOMLY SELECTED VALUE FOR
180 REM      THE 'HEIGHT' OF A TRIANGLE
190 REM  P - NUMBER OF PROBLEMS SELECTED BY USER
200 REM  S - USER ANSWER (VIA INPUT)
210 REM  =============
220 REM
230 PRINT "♡"
240 PRINT, "DRILL ON CALCULATING THE"
250 PRINT, "AREA OF A TRIANGLE"
260 PRINT
270 PRINT "HOW MANY PROBLEMS DO YOU WANT (1-10)";
```

```
280 INPUT P
290 IF P<1 THEN 270
300 IF P<11 THEN 350
310 PRINT "THAT'S TOO MANY...KEEP"
320 PRINT "   IT TO 10 OR LESS."
330 GOTO 260
340 REM  ===USER GOT TO DEFINE THE VALUE OF P (WITHIN LIMITS)
350 FOR I=1 TO P
360 PRINT
370 REM  ===============
380 REM  GET RANDOM VALUES FOR THE BASE AND
390 REM  HEIGHT AND CALCULATE THE AREA.
400 REM  ===============
410 B = INT(10 * RND(0) + 1) * 5
420 H = INT(15 * RND(0) + 1) * 10
430 A = .5 * B * H
440 PRINT "THE BASE OF A TRIANGLE"
450 PRINT "IS" B "INCHES AND ITS HEIGHT"
460 PRINT "IS" H "INCHES. WHAT IS"
470 PRINT "ITS AREA IN SQUARE INCHES";
480 INPUT S
490 REM  ===IS INPUT CORRECT ANSWER?===
500 IF A = S THEN 570
510 PRINT
520 PRINT "NO, AREA = 1/2 x BASE x HEIGHT"
530 PRINT : PRINT "= 1/2 x " B " x " H
540 PRINT : PRINT "=" A "SQUARE INCHES"
550 REM  ===USER CONTROLS WHEN TO GO===
560 PRINT : INPUT "PRESS RETURN TO CONTINUE";Z$ : GOTO 610
570 PRINT "♡":FOR J=1 TO 11:PRINT:NEXT J
580 PRINT,"P E R F E C T!":FOR J=1 TO 500:NEXT J
590 C = C + 1
600 REM  ===CLEAR THE SCREEN THEN GO===
610 PRINT "♡"
620 NEXT I
630 FOR J=1 TO 11:PRINT:NEXT J
640 PRINT,"YOU WERE ASKED" P "PROBLEMS."
650 PRINT : PRINT,"YOU GOT" C "CORRECT."
660 REM ===COMPUTE THE PERCENTAGE SCORE===
670 S = INT((C * 100 / P) * 10)/10
680 PRINT : PRINT, "THAT'S" S "PERCENT!"
690 END

 RUN
DRILL ON CALCULATING THE
AREA OF A TRIANGLE

HOW MANY PROBLEMS DO YOU WANT 1-10)?3
```

```
THE BASE OF A TRIANGLE IS
40 INCHES AND ITS HEIGHT
IS 120 INCHES.  WHAT IS
ITS AREA IN SQUARE INCHES?4800

NO, AREA = 1/2 BASE x HEIGHT
= 1/2 x 40 x 120
= 2400 SQUARE INCHES
PRESS RETURN TO CONTINUE?

THE BASE OF A TRIANGLE IS
5 INCHES AND ITS HEIGHT
IS 130 INCHES.  WHAT IS
ITS AREA IN SQUARE INCHES?325

      P E R F E C T!

THE BASE OF A TRIANGLE IS
40 INCHES AND ITS HEIGHT
IS 20 INCHES.  WHAT IS
ITS AREA IN SQUARE INCHES?400

      P E R F E C T!

YOU WERE ASKED 3 PROBLEMS.
YOU GOT 2 CORRECT.
THAT'S 66.6 PERCENT!
```

6.5 Posers and Problems

1. Following the display of the Z-scores in Program 10, make additions that will produce a display (and, optionally, a printout) of the "count" of each score entered. That is, if four out of all the scores entered were 100 and two scores were 91, the program will display:

```
Score = 100      Count = 4
Score =  91      Count = 2
     .                .
     .                .
     .                .
    etc.             etc.
```

2. Modify Program 13 to randomly select 10 of 15 possible questions (with appropriate answers) of your own choosing. (Remember, the question and answer may be switched by the program.)
3. Modify Program 14 to ask any five questions of your choosing.

Chapter Seven

More Show and Tell: Tutorial, Simulation, and Testing Examples

"You may speak of love and tenderness and passion, but REAL ecstasy is discovering you DID make a backup copy of your program afterall."
paraphrased from
Blackie Sherrod

"After a little experience, a person realizes that he or she can go to bed at midnight and seldom miss anything."
Tri-County Record

[Remember the above when you get to the "Think About This (for Fun)" in Chapter 9.]

Think About This (for Fun)
In going over his books one day, a bookkeeper for a toy company noticed that the word "balloon" had two *sets of double letters, one following the other. Someplace on this page there is a word that has* three *sets of double letters, one right after the other. Can you find it?*

Think About This (Seriously)
Should teachers have the ability to develop instructional computing materials for their own use in the classroom, or should most rely only on "commercially available" software?

7.1 Objectives

For the successful completion of this chapter, you should be able to:

1. Describe the purpose and application of instructional computing programs that are
   a. tutorial (dialog)
   b. simulations, and
   c. testing (Sections 7.2–7.4).
2. Modify certain programs in this chapter to present information of your choosing.
3. Describe in outline form the sequence of execution of each of the example programs in this chapter.

7.2 Tutorial (Dialog) Applications

An extension of the drill-and-practice application allows for more feedback to the user whenever difficulty is indicated. This "tutorial dialog" could assist the user in locating the specific cause of errors, provide hints, or, if needed, branch to a separate section for detailed review.

From an instructional computing standpoint, programs of this type are often the most complicated to design, are time-consuming in development, and generally go through many stages of testing and revision. The reason is that these programs (if carefully and thoroughly designed) must anticipate a variety of user's responses and treat them accordingly: Is the user's answer partly correct? Has the user indicated difficulty to the extent that a branch for review is needed? If the user stops in the middle of an interaction, will the program start again at that point for the user? Should the program record the questions/responses for questions missed? Because of these, and other extensive design, development, and evaluation considerations, thorough tutorial dialog programs are not widely available.

The examples that follow are relatively short programs that illustrate some programming strategies for introducing more of a "dialog" into the interaction. They are by no means examples of instructional computing programs with extensive tutorial applications. However, they do show some of the techniques that may be used in programs of this type.

7.2.1 PROGRAM 16: Providing Hints

Providing hints is a simple example of a tutorial program. These hints may be assigned to variables in the same fashion as questions and answers were in Program 14. In the following program, a counter is used so that (arbitrarily) two hints are given before the correct answer is given. (See statements 4060–4080.) An additional option, HINT, is incorporated into the program. Note that different counters are used for questions answered correctly on the first try, after one hint, or after two hints. (See statements 4160–4180.)

RUN from disk and refer to the listing and run of Program 16.

PROGRAM 16

```
10 REM  PROGRAM 16
20 REM  ===============
30 REM  TUTORIAL (HINTS GIVEN)
40 REM  THIS PROGRAM DEMONSTRATES USE OF
50 REM  MULTIPLE SUBROUTINES, HINTS, AND COUNTERS.
60 REM  USER OPTIONS FOR PROGRAM CONTROL
70 REM  ARE ALSO SHOWN.
80 REM  ===============
90 REM  VARIABLE DICTIONARY
100 REM ===============
110 REM A$  - THE CORRECT ANSWER
120 REM C1  - COUNTER FOR CORRECT 1ST TRY
130 REM C2  - COUNTER FOR CORRECT 2ND TRY
132 REM C3  - COUNTER FOR CORRECT 3RD TRY
140 REM H1$,H2$ - HINTS GIVEN TO MISSED QUESTIONS
150 REM P$()- LIST OF POSITIVE REINFORCERS
160 REM R   - A RANDOM NUMBER (4-1)
170 REM R$  - USER'S RESPONSE
180 REM S$  - DASHES REPRESENTING CORRECT ANSWER
190 REM ===============
200 REM DIMENSION FEEDBACK LIST; ASSIGN VALUES
210 REM ===============
220 DIM P$(4)
230 DATA "GREAT","WONDERFUL","HOT-DOG","THAT'S IT"
240 FOR I = 1 TO 4 : READ P$(I) : NEXT I
250 REM ===============
260 REM SUBROUTINE 1000 FOR SCREEN POSITIONING
270 REM ===============
280 PRINT "♡" : GOSUB 1000
290 PRINT " A  B I T  O F  H I S T O R Y"
300 REM ===============
310 REM SUBROUTINE 2000 FOR PAUSING
320 REM ===============
330 GOSUB 2000
340 REM ===============
350 REM SUBROUTINE 3000 FOR PRESENTING USER
360 REM OPTIONS AT TOP OF SCREEN
370 REM ===============
380 GOSUB 3000
390 GOSUB 1000
400 REM ===============
410 REM SEQUENCE FOR ALL QUESTIONS IS:
420 REM    1. PRINT(S) TO ASK THE QUESTION
430 REM    2. CORRECT ANSWER ASSIGNED TO A$
440 REM    3. DASHES REPRESENTING ANSWER ASSIGNED TO S$
450 REM    4. HINTS ASSIGNED TO H1$,H2$
460 REM    5. GOSUB 4000 FOR INPUT AND ANSWER CHECKING
470 REM ===============
480 PRINT "AMERICA'S FIRST PRESIDENT WAS NAMED:"
490 A$ = "GEORGE WASHINGTON"
500 S$ = "------ ----------"
510 H1$ = "A DOLLAR FOR A SCHOLAR" : H2$ = "GEORGIE-PORGIE"
520 REM ===============
530 REM SUBROUTINE 4000 FOR INPUT AND ANSWER CHECKING
540 REM ===============
550 GOSUB 4000
560 GOSUB 3000
570 GOSUB 1000
580 PRINT "AS A YOUNG LAD, HE CHOPPED DOWN A:"
590 A$ = "CHERRY TREE" : S$ = "------ ----"
592 H1$ = "PITS" : H2$ = "PIES"
600 GOSUB 4000
610 GOSUB 3000
620 GOSUB 1000
630 PRINT "HOWEVER, HE WAS CAUGHT RED-HANDED (NO"
640 PRINT "PUN INTENDED) AND COULD NOT TELL A:"
650 A$ = "LIE" : S$ = "---"
652 H1$ = "FIB" : H2$ = "-?-, LAY, LAID"
660 GOSUB 4000
670 REM ===ROOM FOR MORE QUESTIONS===
960 PRINT "♡" : GOSUB 1000
962 PRINT "YOU ANSWERED" C1 + C2 + C3 "QUESTION(S)." : PRINT
970 PRINT "YOU ANSWERED" C1 "ON THE FIRST TRY." : PRINT
980 PRINT "YOU ANSWERED" C2 "ON THE SECOND TRY." : PRINT
982 PRINT "YOU ANSWERED" C3 "ON THE THIRD TRY."
990 END
999 REM ===SCREEN POSITIONING SUBROUTINE===
1000 FOR I = 1 TO 8 : PRINT : NEXT I
1010 RETURN
1999 REM ===PAUSING SUBROUTINE===
2000 FOR I = 1 TO 1000 : NEXT I
2010 RETURN
2999 REM ===OPTION DISPLAY SUBROUTINE===
3000 PRINT "♡"
3010 PRINT,"INPUT OPTIONS:"
3020 PRINT "'HINT'  'ANSWER'  'SKIP'  'STOP'"
3030 RETURN
```

```
3999 REM ===INPUT AND CHECKING SUBROUTINE===
4000 PRINT TAB(12) S$
4010 PRINT TAB(10) : INPUT R$
4012 IF R$ = "HINT" THEN 4060
4020 IF R$ = "ANSWER" THEN 4070
4030 IF R$ = "SKIP" THEN 4190
4040 IF R$ = "STOP" THEN PRINT "♡" : GOTO 960
4050 IF R$ = A$ THEN 4120
4060 PRINT : F = F + 1 : ON F GOTO 4062,4064,4070
4062 PRINT "HERE'S A HINT: " H1$ : GOTO 4000
4064 PRINT "HERE'S ANOTHER HINT: " H2$ : GOTO 4000
4070 PRINT "A CORRECT ANSWER IS " A$ : PRINT
4080 INPUT "PRESS RETURN TO CONTINUE";Z$ : GOTO 4190
4090 REM ===============
4100 REM CORRECT ANSWER PROCEDURE
4110 REM ===============
4120 PRINT "♡" : GOSUB 1000
4130 R = INT(4 * RND(0) + 1)
4140 PRINT TAB(15) P$(R)"!"
4150 GOSUB 2000
4160 IF F = 0 THEN C1 = C1 + 1
4170 IF F = 1 THEN C2 = C2 + 1
4180 IF F = 2 THEN C3 = C3 + 1
4190 REM ===SET F TO ZERO BEFORE NEXT QUESTION===
4200 F = 0 : RETURN
```

```
RUN
A  B I T  O F  H I S T O R Y


         INPUT OPTIONS:
'HINT'  'ANSWER'  'SKIP'  'STOP'

AMERICA'S FIRST PRESIDENT WAS NAMED:
           ------ ----------
?HINT
HERE'S A HINT: A DOLLAR FOR A SCHOLAR
           ------ ----------
?JACSON
HERE'S ANOTHER HINT: GEORGIE-PORGIE
           ------ ----------
?GEORGE WASHINGTON

          THAT'S IT!


         INPUT OPTIONS:
'HINT'  'ANSWER'  'SKIP'  'STOP'

AS A YOUNG LAD, HE CHOPPED DOWN A:
           ------ ----
```

```
?SKIP

         INPUT OPTIONS:
'HINT'  'ANSWER'  'SKIP'  'STOP'
HOWEVER, HE WAS CAUGHT RED-HANDED (NO
PUN INTENDED) AND COULD NOT TELL A:
          ---
          ?STORY
HERE'S A HINT: FIB
          ---
?LIE

          WONDERFUL!

YOU ANSWERED 2 QUESTION(S).
YOU ANSWERED 0 ON THE FIRST TRY.
YOU ANSWERED 1 ON THE SECOND TRY.
YOU ANSWERED 1 ON THE THIRD TRY.
```

7.2.2 PROGRAM 17: Review of Missed Questions

Another approach for providing hints is to assign questions, answers, and hints to one-dimensional arrays. Questions may be randomly selected and the appropriate hint given on the first miss. Program 17 follows this approach, but also "flags" each question missed by storing the user's incorrect response in another array (see statements 554–560). At the conclusion of the program, a review of missed questions can be provided by checking this array to see if it contains a user's response (see statements 760–810).

Also note the method to determine if *all* questions were answered correctly on the *first* try (statement 690) and the method for computing a percentage score (statement 830).

RUN from disk and refer to the listing and run of Program 17.

PROGRAM 17

```
10 REM  PROGRAM 17
20 REM  ===============
30 REM  TUTORIAL (HINTS GIVEN)
40 REM  THIS PROGRAM DEMONSTRATES STORING
50 REM  QUESTIONS, ANSWERS, AND HINTS IN
60 REM  ONE-DIMENSIONAL ARRAYS (LISTS).
70 REM  QUESTIONS ARE RANDOMLY SELECTED FROM
80 REM  THE LIST WITHOUT REPETITION.
90 REM  THE USER'S RESPONSE TO A MISSED
100 REM QUESTION IS STORED IN ANOTHER ARRAY
110 REM FOR REVIEW AT THE END OF THE SESSION.
120 REM ===============
130 REM VARIABLE DICTIONARY
```

```
140 REM ===============
150 REM A$() - A LIST OF ANSWERS
150 REM C    - COUNTER FOR CORRECT 1ST TRY
170 REM C$() - A LIST OF POSITIVE FEEDBACK
180 REM F    - FLAG INDICATING A MISSED QUESTION
190 REM H$() - A LIST OF HINTS
200 REM Q    - THE NUMBER OF QUESTIONS PRESENTED
210 REM R    - A RANDOM NUMBER (3-1)
212 REM R$   - THE USER'S RESPONSE
220 REM S$() - A LIST OF INCORRECT RESPONSES
230 REM W$() - A LIST OF FEEDBACK FOR INCORRECT ANSWERS
240 REM X    - A RANDOM NUMBER (10-1)
250 REM        FOR QUESTION SELECTION
280 REM Z(X) - FLAG FOR A SELECTED QUESTION IN THE LIST
270 REM
280 REM ===DIMENSION ALL ARRAYS===
290 REM
300 DIM Q$(10),A$(10),H$(10),S$(10),Z(10),C$(3),N$(3)
310 REM
320 REM ===STORE FEEDBACK LISTS===
330 REM
340 FOR I = 1 TO 3 : READ C$(I),W$(I) : NEXT I
350 REM
360 REM ===STORE QUESTION, ANSWER, HINT IN LISTS===
370 REM
380 FOR I = 1 TO 10 : READ Q$(I),A$(I),H$(I) : NEXT I
390 REM
400 REM ===SHOW TITLE (INTRODUCTION, ETC.)===
410 REM
420 GOSUB 5000
430 PRINT, "FUN AND GAMES"
440 GOSUB 6000
450 GOSUB 5000
460 PRINT, "I HAVE 10 QUESTIONS." : PRINT,
470 INPUT "HOW MANY WOULD YOU LIKE";Q
480 IF Q < 1 OR Q > 10 THEN 450
490 FOR P = 1 TO Q
500 GOSUB 5000
510 X = INT(10 * RND(0) + 1)
520 IF Z(X) = 1 THEN 510
522 REM
524 REM ===FLAG LIST ITEM; GET RANDOM VALUE FOR FEEDBACK===
526 REM
530 Z(X) = 1 : R = INT(3 * RND(0) + 1)
540 PRINT : PRINT Q$(X); : INPUT R$
550 IF R$ = A$(X) THEN 590
552 REM
554 REM ===ASSIGN INCORRECT ANSWER TO S$(X)===
556 REM
```

```
560 S$(X) = R$
562 IF F = 0 THEN F = 1 : PRINT,W$(R) : PRINT,H$(X) : GOTO
    540
570 PRINT : PRINT "A CORRECT ANSWER IS:" : PRINT, A$(X)
580 PRINT : INPUT "PRESS RETURN TO CONTINUE";Z$ : GOTO 630
582 REM
584 REM ===CORRECT ANSWER GIVEN PROCEDURE===
586 REM
590 GOSUB 5000 : PRINT, C$(R)"!"
610 IF F = 0 THEN C = C + 1
620 GOSUB 6000
630 F = 0
640 NEXT P
650 REM
660 REM ===SHOW QUESTIONS MISSED IF ANY===
670 REM
680 GOSUB 5000
690 IF C = Q THEN PRINT,"EXCELLENT WORK!" : PRINT : GOTO 830
700 PRINT,"HERE'S A REVIEW..." : GOSUB 6000
710 FOR J = 1 TO 10
720 REM
730 REM ===IF S$(J) IS NULL ("") THEN OK===
740 REM
750 IF S$(J) = "" THEN 810
760 GOSUB 5000
770 PRINT,"QUESTION" : PRINT Q$(J) : PRINT
780 PRINT,"YOUR ANSWER" : PRINT,S$(J) : PRINT
790 PRINT,"A CORRECT ANSWER" : PRINT,A$(J) : PRINT
800 INPUT "PRESS RETURN TO CONTINUE...";Z$
810 NEXT J
820 GOSUB 5000
830 PRINT "YOUR SCORE IS" INT((C * 100/Q) * 10)/10
    "PERCENT!"
840 END
850 REM
860 REM ===DATA FOR FEEDBACK===
870 REM
880 DATA "HOT-DOGGIES","NO, LET ME HELP"
890 DATA "SWELL","WILL THIS HELP:"
900 DATA "YOU GOT IT","HERE'S A HINT"
910 REM
920 REM ===DATA FOR QUESTION, ANSWER, HINT===
930 REM
940 DATA "LAST NAME OF 'BOLERO' COMPOSER","RAVEL","SWEATERS
    CAN UN-"
950 DATA "LONGEST RIVER IN THE WORLD","NILE","ASP ME NO
    QUESTIONS"
960 DATA "LARGEST RIVER IN THE WORLD","AMAZON","BIG-MOMMA!"
970 DATA "THIS PROGRAM IS WRITTEN IN","BASIC","FUNDAMENTAL"
```

```
990 DATA "MONTH OF SHORTEST DAY IN TEXAS","DECEMBER","MAKE
    MERRY"
990 DATA "DANIEL WAS PLACED IN THE LION'S","DEN","FAMILY
    ROOM"
1000 DATA "LARGEST OF THE GREAT LAKES","SUPERIOR","SUPER-
    BIG"
1010 DATA "MARY HAD A LITTLE","LAMB","FLEECED ON THIS ONE?"
1020 DATA "39.37 INCHES = ONE","METER","IS YOURS RUNNING?"
1030 DATA "JACK AND JILL WENT UP THE","HILL","NO MOUNTAIN
    THIS"
4997 REM
4998 REM ===POSITIONING SUBROUTINE===
4999 REM
5000 PRINT "♡" : FOR I = 1 to 9 : PRINT : NEXT I
5010 RETURN
5997 REM
5998 REM ===PAUSING SUBROUTINE===
5999 REM
6000 FOR I = 1 TO 500 : NEXT I
6010 RETURN

RUN
        FUN AND GAMES

I HAVE 10 QUESTIONS.
HOW MANY WOULD YOU LIKE?4

GOLIATH'S SLAYER?DAVID

         HOT-DOGGIES!

MONTH OF THE LONGEST DAY?DECEMBER
WHOA NOW...! HERE'S A HINT:
ASSUME N. HEMISPHERE
MONTH OF THE LONGEST DAY?JUNE

         HOT-DOGGIES!

WHICH IS LONGER: METER OR YARD?METER

         GREAT!

LONGEST RIVER IN THE WORLD?AMAZON
WHOA NOW...! HERE'S A HINT:
A SHADE OF GREEN
LONGEST RIVER IN THE WORLD?MISSISSIPPI
A CORRECT ANSWER IS: NILE

PRESS RETURN TO CONTINUE
```

```
YOU ANSWERED 2 CORRECTLY
ON THE FIRST TRY...

         HERE'S A REVIEW...

     QUESTION:
MONTH OF THE LONGEST DAY

     YOUR ANSWER:
DECEMBER

     A CORRECT ANSWER:
JUNE

     PRESS RETURN TO CONTINUE

     QUESTION:
LONGEST RIVER IN THE WORLD

     YOUR ANSWER:
AMAZON

     A CORRECT ANSWER:
NILE

     PRESS RETURN TO CONTINUE

YOUR SCORE IS 50 %!
```

7.2.3 PROGRAM 18: Random Selection of "Lengthy" Questions

Program 17 demonstrated one approach to randomly selecting questions from an array. However, all of its questions were "one-liners." It is often the case that one line of output is not sufficient for presenting information, examples, and so forth *before* asking a given question.

Program 16 demonstrated one approach to asking *linear* questions, providing hints, and so on, where more than one line of output was possible. But how can we *randomly* select the questions?

One technique is simply to combine the strategies of these two programs. That is, we will use as many PRINT statements as needed to present the question, then assign hints and the correct answer(s) to variable(s), but have all possible questions within a FOR-NEXT loop (see statements 500–5110). Within the loop, a random number generator (statement 520) will be used to give some number within the range of possible questions available. Random numbers selected will be "flagged" to avoid repetition as we've done before. Using these random numbers, an ON-GOTO (statement 570) will transfer execution to the appropriate line number that begins presentation of the question, assigns hints and answers, and so forth. This will be followed by a GOTO (statements 650–670) that transfers execution to the "answer-checking" portion of the loop.

Program 18 is a *skeletal* program that illustrates this technique. Can you modify and complete this program to include an introduction, present more realistic questions, and give a more detailed user "performance report"?

RUN from disk and refer to the listing and run of Program 18.

PROGRAM 18

```
10 REM  PROGRAM 18
20 REM  ===============
30 REM  TUTORIAL ('LENGTHY' QUESTIONS WITH HINTS)
40 REM  THIS PROGRAM DEMONSTRATES HOW QUESTIONS
50 REM  OF MORE THAN ONE LINE IN LENGTH MAY BE
60 REM  RANDOMLY SELECTED.  PRINT STATEMENTS
70 REM  USED TO PRESENT THE QUESTION. THE CORRECT
80 REM  ANSWER AND HINT, HOWEVER, ARE ASSIGNED TO
90 REM  ARRAYS.  A RANDOM NUMBER BETWEEN 1 AND
100 REM THE MAXIMUM NUMBER OF QUESTIONS WILL
110 REM DEFINE THE BEGINNING LINE NUMBER OF THE
120 REM SELECTED QUESTION THROUGH USE OF AN
130 REM ON-GOTO STATEMENT.  THIS NUMBER WILL ALSO
140 REM DEFINE THE CORRECT ANSWER AND HINT.
150 REM ===============
160 REM VARIABLE DICTIONARY
170 REM ===============
180 REM A$() - A LIST OF ANSWERS
190 REM H$() - A LIST OF HINTS
200 REM F    - A FLAG TO EITHER GIVE A HINT (0) OR ANSWER
    (1)
210 REM Q    - A QUESTION LOOP COUNTER
220 REM R$   - A USER'S RESPONSE
230 REM X    - A RANDOM NUMBER (5-1)
240 REM Z(X) - A FLAG FOR A SELECTED NUMBER
250 REM ===============
260 REM
270 REM ===DIMENSION LISTS TO MAXIMUM SIZE===
280 REM
290 DIM A$(5),H$(5),Z(5)
300 REM
310 REM ===STORE ANSWERS AND HINTS===
320 REM
330 FOR I = 1 TO 5 : READ A$(I),H$(I) : NEXT I
340 REM
```

```
350 REM ===TITLE, INTRODUCTION, ETC. HERE===
360 REM
370 PRINT "♡" : FOR I = 1 TO 10 : PRINT : NEXT I
380 PRINT "    BASIC (AS IN COMPUTER) REVIEW"
390 FOR I = 1 TO 3000 : NEXT I : PRINT "♡"
400 PRINT : PRINT "THIS PROGRAM FRAGMENT DEMONSTRATES HOW"
410 PRINT : PRINT "QUESTIONS LONGER THAN ONE LINE MAY BE"
420 PRINT : PRINT "ASKED AT RANDOM WITHOUT REPETITION.  5"
430 PRINT : PRINT "QUESTIONS ARE POSSIBLE IN THIS EXAMPLE."
440 PRINT : INPUT "HOW MANY DO YOU WANT TO TRY";Q
460 IF Q < 1 OR Q > 5 THEN 440
470 REM
480 REM ===QUESTION LOOP===
490 REM
500 FOR I = 1 TO Q
510 PRINT "♡"
520 X = INT(5  * RND(0) + 1)
530 IF Z(X) = 1 THEN 520
540 REM
550 REM ===FLAG NUMBER SELECTED; GOTO QUESTION LINE===
580 REM
570 Z(X) = 1 : ON X GOTO 610,710,780,810,840
580 REM
590 REM ===FIRST QUESTION===
600 REM
810 PRINT "THE BASIC STATEMENT THAT IS USED TO"
620 PRINT "DISPLAY TEXT OR VARIABLE VALUES ON"
630 PRINT "THE SCREEN OR PRINTER IS";
840 REM
650 REM ===AFTER PRINTING THE QUESTION, GOTO ANSWER CHECK===
660 REM
670 GOTO 5000
680 REM
690 REM ===NEXT QUESTION===
700 REM
710 PRINT "THE MAXIMUM LINE LENGTH FOR A BASIC"
720 PRINT "STATEMENT ON THE COMMODORE 64 MICRO-"
730 PRINT "COMPUTER IS -?- CHARACTERS";
740 GOTO 5000
750 REM
760 REM ===SEQUENCE FOR REMAINING QUESTIONS SAME AS ABOVE===
770 REM
780 PRINT "THE BASIC FUNCTION TO CHANGE A DECIMAL"
790 PRINT "VALUE TO AN INTEGER IS";
800 GOTO 5000
810 PRINT "WHAT STATEMENT MUST BE AT THE END OF A"
820 PRINT "BASIC SUBROUTINE";
830 GOTO 5000
840 PRINT "THE BASIC STATEMENT THAT REQUIRES"
```

```
850 PRINT "DEPRESSING THE 'RETURN' KEY IS";
860 GOTO 5000
870 REM ===============
880 REM THERE'S ROOM TO ADD MORE QUESTIONS HERE.
890 REM THE '5' IN STATEMENTS 290,330,420,480 AND 520
900 REM MUST BE CHANGED TO THE APPROPRIATE NUMBER,
910 REM AND ANSWER, HINT DATA MUST BE ADDED.
920 REM APPROPRIATE LINE NUMBERS MUST BE ADDED
930 REM TO STATEMENT 570.
940 REM ===============
5000 INPUT R$ : PRINT
5010 IF R$ = A$(X) THEN 5080
5020 IF F=0 THEN F=1:PRINT "HINT: " H$(X):PRINT
     "ANSWER";:GOTO 5000
5030 PRINT:PRINT "A CORRECT ANSWER IS " A$(X)
5040 PRINT : INPUT "PRESS RETURN TO CONTINUE";Z$ : GOTO 5100
5050 REM
5060 REM ===CORRECT ANSWER PROCEDURE===
5070 REM
5080 PRINT "♡" : FOR P = 1 TO 10 : PRINT : NEXT P :
     PRINT,,"OKEY-DOKEY"
5090 FOR P = 1 TO 500 : NEXT P
5100 F = 0
5110 NEXT I
5120 REM
5130 REM ===QUESTIONS COMPLETED===
5140 REM
5150 PRINT "♡" : FOR P = 1 TO 12 : PRINT : NEXT P
5160 PRINT "CAN YOU ADD A CONCLUSION TO THIS?"
5170 END
5970 REM
5980 REM ===DATA FOR ANSWER AND HINT FOR QUESTIONS===
5990 REM
6000 DATA "PRINT","MIGHTIER THAN THE SWORD"
6010 DATA "80","SQR(6400)"
6020 DATA "INT","VERY INTERESTING"
6030 DATA "RETURN","-?- TO SENDER"
6040 DATA "INPUT","'?'"

RUN
    BASIC (AS IN COMPUTER) REVIEW
THIS PROGRAM FRAGMENT DEMONSTRATES HOW
QUESTIONS LONGER THAN ONE LINE MAY BE
ASKED AT RANDOM WITHOUT REPETITION. 5
QUESTIONS ARE POSSIBLE IN THIS EXAMPLE.
HOW MANY DO YOU WANT TO TRY?10
HOW MANY DO YOU WANT TO TRY?3
```

```
THE BASIC STATEMENT THAT REQUIRES
DEPRESSING THE 'RETURN' KEY IS?GOSUB

HINT: '?'
YOUR ANSWER?INPUT

         OKEY-DOKEY
THE BASIC FUNCTION TO CHANGE A DECIMAL
VALUE TO AN INTEGER IS?INT

         OKEY-DOKEY
WHAT STATEMENT MUST BE AT THE END OF A
BASIC SUBROUTINE?END

HINT: -?- TO SENDER
YOUR ANSWER?SEND

A CORRECT ANSWER IS RETURN

PRESS RETURN TO CONTINUE

CAN YOU ADD A CONCLUSION TO THIS?

RUN
    BASIC (AS IN COMPUTER) REVIEW
THIS PROGRAM FRAGMENT DEMONSTRATES HOW
QUESTIONS LONGER THAN ONE LINE MAY BE
ASKED AT RANDOM WITHOUT REPETITION. 5
QUESTIONS ARE POSSIBLE IN THIS EXAMPLE.

HOW MANY DO YOU WANT TO TRY?3
WHAT STATEMENT MUST BE AT THE END OF A
BASIC SUBROUTINE?RETURN

         OKEY-DOKEY
THE BASIC STATEMENT THAT IS USED TO
DISPLAY TEXT OR VARIABLE VALUES ON
THE SCREEN OR PRINTER IS?PRINT

         OKEY-DOKEY
THE MAXIMUM LINE LENGTH FOR A BASIC
STATEMENT ON THE COMMODORE 64 MICRO-
COMPUTER IS -?- CHARACTERS?80

         OKEY-DOKEY

CAN YOU ADD A CONCLUSION TO THIS?
```

7.2.4 PROGRAM 19: Step-by-Step Dialog

A tutorial program can do more than just give hints when users are having difficulty with a given question. It can, to some degree, approach the type of dialog that occurs between a tutor and a student. As an example, consider a question related to the chemical concept of a *mole*. By definition, a mole is a quantity of a chemical compound equal to the formula weight (FW) of that compound. This quantity is usually expressed in grams, but it could be any mass unit (ounces, tons, etc.). For a given weight of a chemical compound, the number of moles is determined by the following formula:

$$\text{Moles} = \text{weight (grams)} / \text{FW (gram-formula weight)}$$

The basis for this type of dialog example is in defining the main concept as a series of steps (or subconcepts) that lead to the correct solution. For this chemical concept, these steps are simply:

1. Was the correct formula used?
2. Were the correct values applied to the formula?
3. If steps 1 and 2 are true, yet the user missed the question, then a "math error" must have occurred.

Tutorial programs of this nature may be written by defining any concept into a step-by-step approach for solution or explanation. The following program illustrates a type of dialog that could occur in a tutorial instructional computing application. Note that the program makes use of the ABS (absolute) function to allow for a tolerance of + or − 0.1 in the student's answers (see statements 450–500 and 680). Also be reminded that this is, in essence, a program fragment and does not include an introduction, examples, random selection of positive responses, use of counters for the number correct, and so on. These elements should always be incorporated in programs for actual use in an educational setting.

RUN from disk and refer to the listing and run of Program 19.

PROGRAM 19

```
10 REM  PROGRAM 19
20 REM  ============
30 REM  TUTORIAL 'DIALOG'
40 REM  PROGRAM DEMOS MORE OF A 'TUTORIAL'
50 REM  TYPE OF INTERACTION BETWEEN USER AND
60 REM  THE PROGRAM, USING THE CHEMICAL CONCEPT
70 REM  OF THE 'MOLE' AS AN ILLUSTRATIVE VEHICLE.
80 REM  ONLY THREE COMPOUNDS ARE USED IN
90 REM  THE EXAMPLE WITH THEIR FORMULAS AND
100 REM  FORMULA WEIGHTS STORED IN LISTS.
110 REM  USE OF THE ABS (ABSOLUTE) FUNCTION
```

```
120 REM  IS ALSO INTRODUCED.
130 REM  ============
140 REM  VARIABLE DICTIONARY
150 REM  ============
160 REM  C$() - CHEMICAL COMPOUND FORMULA
170 REM  F     - FLAG FOR MISSING QUESTION 1ST TRY
180 REM  G     - RANDOM NUMBER OF GRAMS OF COMPOUND
190 REM  M     - NUMBER OF MOLES (GRAMS/FORMULA WT)
200 REM  R,R$,V - USER RESPONSES (VIA INPUTS)
210 REM  W()  - FORMULA WEIGHT OF A COMPOUND
220 REM  X     - RANDOM NUMBER (3-1) FOR
230 REM          COMPOUND SELECTION
240 REM  =================
250 DIM C$(3),W(3)
260 DATA "KOH",56,"HF",20,"KI",166
270 REM  ===STORE THE FORMULAS AND WEIGHTS===
280 FOR I=1 TO 3
280 READ C$(I),W(I)
300 NEXT I
310 PRINT "♡" : F = 0
320 REM  =================
330 REM  GET A RANDOM NUMBER OF GRAMS AND A
340 REM  RANDOM COMPOUND. THEN CALCULATE MOLES.
350 REM  ===============
360 G = INT(10 * RND(0) + 1) * 20
370 X = INT(3 * RND(0) + 1)
380 M = G / W(X)
390 REM  ===ASK THE QUESTION===
400 PRINT
410 PRINT"HOW MANY MOLES OF " C$(X) " ARE"
420 PRINT"PRESENT IN" G "GRAMS";
430 INPUT 4
440 PRINT
450 REM  ===============
460 REM  USE THE ABS FUNCTION TO ACCEPT AN
470 REM  ANSWER THAT IS WITHIN 0.1 OF THE
480 REM  CORRECT ANSWER AND THE USER'S ANSWER.
490 REM  ================
500 IF ABS(R-M) <= .1 THEN 860
510 REM  ===GIVE ANSWER ON SECOND MISS===
520 IF F=1 THEN 800
530 F=1
540 REM ===ASK FIRST STEP IN SOLUTION SEQUENCE===
550 PRINT "NO...DID YOU DIVIDE THE"
560 PRINT "WEIGHT BY THE FW (Y OR N)";
570 INPUT R$
580 PRINT
580 IF R$ = "Y" THEN 630
600 PRINT"WELL, YOU SHOULD! TRY AGAIN."
```

```
610 GOTO 400
620 REM  ===CHECK FOR SECOND STOP IN SOL'N SEQUENCE===
630 PRINT "GOOD, THAT IS CORRECT."
640 PRINT "WHAT VALUE DID YOU USE"
650 PRINT "FOR THE FW OF " C$(X);
660 INPUT V
670 PRINT
680 IF ABS(V - W(X)) <= .1 THEN 750
690 PRINT"AHA!  THIS MAY BE YOUR"
700 PRINT "PROBLEM.  THE APPROXIMATE"
710 PRINT "FW OF " C$(X) " IS" W(X)
720 PRINT"NOW TRY IT AGAIN..."
730 GOTO 400
740 REM  ===IF CORRECT TO HERE, MUST BE MATH ERROR===
750 PRINT"HMMM...THAT IS THE CORRECT"
760 PRINT "FW FOR " C$(X) ", YOU MUST"
770 PRINT"HAVE MADE AN ARITHMETIC"
780 PRINT"ERROR.  CHECK AND TRY AGAIN."
790 GOTO 400
800 PRINT
810 REM  ===SHOW THE CORRECT SOLUTION===
820 PRINT"MOLES = WT/FW"
830 PRINT "=" G "/" W(X)
840 PRINT "=" INT(M * 100)/100
850 GOTO 880
860 PRINT"   E X C E L L E N T !"
870 REM  ===LET USER CONTINUE AT WILL===
880 PRINT
890 INPUT "WANT ANOTHER (Y OR N)";R$
900 IF R$ = "Y" THEN 310
910 END

RUN

HOW MANY MOLES OF KOH ARE
PRESENT IN 180 GRAMS?.5

NO...DID YOU DIVIDE THE
WEIGHT BY THE FW (Y OR N)?Y

GOOD. THAT IS CORRECT.
WHAT VALUE DID YOU USE
FOR THE FW OF KOH?56

HMMM...THAT IS THE CORRECT
FW FOR KOH. YOU MUST
HAVE MADE AN ARITHMETIC
ERROR.  CHECK AND TRY AGAIN.
```

```
HOW MANY MOLES OF KOH ARE
PRESENT IN 180 GRAMS?3

MOLES = WT/FW
= 180/56
= 3.21

WANT ANOTHER (Y OR N)?Y

HOW MANY MOLES OF KI ARE
PRESENT IN 120 GRAMS?.8

   E X C E L L E N T !

WANT ANOTHER (Y OR N)?Y

HOW MANY MOLES OF KI ARE
PRESENT IN 100 GRAMS?2

NO...DID YOU DIVIDE THE
WEIGHT BY THE FW (Y OR N)?Y

GOOD. THAT IS CORRECT.
WHAT VALUE DID YOU USE
FOR THE FW OF KI?55

AHA!  THIS MAY BE YOUR
PROBLEM.  THE APPROXIMATE
FW OF KI IS 165
NOW TRY IT AGAIN...

HOW MANY MOLES OF KI ARE
PRESENT IN 100 GRAMS?.6

   E X C E L L E N T !

WANT ANOTHER (Y OR N)?N
```

7.3 Simulation Applications

Usually, simulation applications in instructional computing are used when it is important to understand a given concept, but one or more of the following situations apply:

1. There are time and/or space and/or equipment limitations.
2. The real process might place the user in a perilous situation.

3. Review and/or practice would be beneficial prior to performing the actual experiment or process.

Simulations in instructional computing are based upon models, most of which are mathematical in origin. In general, these programs allow a user to manipulate parameters and perhaps discover the effect of those manipulations. One popular application is in population dynamics: What happens to the population over a certain span of years if both the birth and death rates decrease and the female/male birth ratio increases? Another application is in environmental studies: What happens to the water oxygen content if untreated raw sewage is dumped into a slow-moving stream? A fast-moving river? What is the effect of performing primary treatment? Secondary treatment? How does temperature affect the above results?

Again, the example programs discussed below are not extensive simulations; they do illustrate the concept of basing the design on some defined model.

7.3.1 PROGRAM 20: Rolling a Pair of Dice

One easy model to simulate is rolling a pair of dice. Any die rolled will give a random number, 1 through 6. The sum of the two dice is the roll value. Given an *infinite* number of rolls, what number is cast most often? What would be your guess? Program 20 can provide a simulated, but, nonetheless, fairly accurate answer to this question (within roll limits!).

Note the technique used in statements 582–586 to determine the value of the number cast most often. Note also how the IF-THEN-PRINT is used to display a response in statements 588–590. (Be sure to note the semicolon at the end of statement 588!)

Remember, the most important point in designing and developing any simulation is in defining the model. Once this is done, it *may* be possible to design a simulation of the model.

RUN from disk and refer to the listing and run of Program 20.

PROGRAM 20

```
10 REM  PROGRAM 20
20 REM  ==============
30 REM  SIMULATION
40 REM  PROGRAM DEMONSTRATES ROLLING DICE
50 REM  UP TO 1000 TIMES, GIVING THE DISTRIBUTION
60 REM  FOR EACH SET OF ROLLS.  EXAMINATION OF
70 REM  THE PERCENTAGE A GIVEN NUMBER IS CAST
80 REM  MAY BE USED TO ILLUSTRATE THE NORMAL
90 REM  DISTRIBUTION CURVE OF THE RANDOM-NUMBER GENERATOR.
110 REM  ==============
120 REM  VARIABLE DICTIONARY
130 REM  ===============
140 REM  D1    - FIRST DIE (OF A PAIR OF DICE)
150 REM  D2    - A SECOND DIE
152 REM  L     - LARGEST NUMBER CAST
154 REM  N     - USER 'GUESS' OF NUMBER MOST OFTEN CAST
160 REM  P()   - COUNT OF A GIVEN VALUE FOR
170 REM          A ROLL OF A PAIR OF DICE
180 REM  P1()  - PERCENTAGE DISTRIBUTION
190 REM  R     - NUMBER OF ROLLS (VIA INPUT)
200 REM  S     - SUM OF D1 AND D2 (VALUE OF A GIVEN ROLL)
210 REM  ===============
220 DIM P(12),P1(12)
230 PRINT "♡"
240 PRINT,"THIS PROGRAM SIMULATES"
250 PRINT,"ROLLING A PAIR OF DICE"
260 REM
270 REM  ===INITIALIZE THE COUNT ARRAY===
260 REM
290 FOR I = 2 TO 12 : P(I) = 0 : NEXT I
300 PRINT
310 INPUT "HOW MANY ROLLS WOULD YOU LIKE";R
320 REM ===CHECK FOR REASONABLE NUMBER OF ROLLS===
330 IF R < 10 OR R >1000 THEN PRINT "I WANT 10 - 1000!" :
    GOTO300
340 PRINT "♡" : FOR I = 1 TO 11 : PRINT : NEXT I
350 PRINT "WHAT NUMBER DO YOU THINK WILL BE CAST"
360 INPUT "MOST OFTEN (2-12)";N
370 IF N < 2 OR N > 12 THEN 340
380 REM  ===SHOW THE USER WE'RE DOING IT===
390 PRINT "♡":FOR I=1 TO 10:PRINT:NEXT I:PRINT,"PATIENCE..."
392 REM
400 REM  ===DO THE ROLLS===
402 REM
410 FOR T=1 TO R
420 REM  ===GET A RANDOM VALUE FOR EACH DIE===
430 D1 = INT(6 * RND(0) + 1)
440 D2 = INT(6 * RND(0) + 1)
450 REM  ===SUM THE PAIR OF DICE===
460 S = D1 + D2
470 REM  ===INCREASE THAT COUNT BY ONE===
480 (P)S = P(S) + 1
490 NEXT T
492 REM
494 REM ===ROLLS COMPLETED===
496 REM
500 PRINT "♡"
510 PRINT "VALUE OF ROLL" TAB(19)"COUNT" TAB(31)"%"
520 PRINT
```

```
540 FOR L=2 TO 12
546 REM
550 REM ===NOTE THE METHOD OF ROUNDING UP===
552 REM
560 P1(L)=INT(((P(L) /R) * 100) + .5)
570 PRINT TAB(6) L TAB(20) P(L) TAB(29) INT((P(L) * 100/R) *
    100)/100
580 NEXT L
581 REM
582 REM ===FIND LARGEST NUMBER CAST===
583 REM
584 B = 0 : FOR I = 2 TO 12 : IF P(I) <= B THEN 586
585 B = P(I) : L = I
586 NEXT I
587 PRINT : PRINT "THE NUMBER CAST MOST OFTEN WAS" L
588 PRINT "YOUR NUMBER WAS" N;
589 IF L = N THEN PRINT "...VERY GOOD!" : GOTO 592
590 PRINT "...TOO BAD!"
592 PRINT
600 PRINT "WANT TO SEE THE DISTRIBUTION"
610 PRINT"CURVE (Y OR N)";
620 INPUT A$
630 IF A$<> "Y" THEN 730
640 PRINT "♡":PRINT"      P E R C E N T A G E"
650 PRINT"    D I S T R I B U T I O N"
652 PRINT"    ----------------------"
660 FOR I=2 TO 12
670 PRINT I;TAB(4);"I ";
680 FOR J=1 TO P1(I)
682 IF P1(I) = 0 THEN 700
684 REM
686 REM ===PRINT OUT GRAPHIC CHARACTER USING ASCII CODE===
688 REM
690 PRINT CHR$(162);
700 NEXT J
710 PRINT
720 NEXT I
722 PRINT "    --------------------------"
724 PRINT "     1234567891111111111222222"
726 PRINT "              0123456789012345"
728 PRINT "          P E R C E N T"
730 PRINT:INPUT "WANT ANOTHER SET OF ROLLS (Y OR N)";Z$
740 IF Z$ = "Y" THEN 270
750 END

RUN
THIS PROGRAM SIMULATES
ROLLING A PAIR OF DICE
```

```
HOW MANY ROLLS WOULD YOU LIKE?500

WHAT NUMBER DO YOU THINK WILL BE CAST
MOST OFTEN (2-12)7

       PATIENCE...NOW ROLLING
VALUE OF ROLL    COUNT       %

     2              14      2.79
     3              25      5
     4              36      7.19
     5              65      13
     6              62      12.39
     7              79      15.8
     8              69      13.8
     9              50      10
     10             50      10
     11             32      6.4
     12             18      3.59

THE NUMBER CAST MOST OFTEN WAS 7
YOUR NUMBER WAS 7...VERY GOOD!

WANT TO SEE THE DISTRIBUTION
CURVE (Y OR N)?Y

     P E R C E N T A G E
   D I S T R I B U T I O N
   -----------------------
2  I **
3  I *****
4  I *******
5  I *************
6  I ************
7  I ***************
8  I *************
9  I **********
10 I **********
11 I ******
12 I ***
    --------------------------
     1234567891111111111222222
              0123456789012345
          P E R C E N T

WANT ANOTHER SET OF ROLLS (Y OR N)?N
```

7.3.2 PROGRAM 21: Dealing a Bridge Hand

As might be expected, one easy model to simulate is a deck of 52 cards. Manipulations, however, are limited to "shuffling" the deck then observing the deal. A simulated deck may be considered as a two-dimensional array of 13 rows (card values) by 4 columns (suits). Two random-number generators can pick (deal) a given row and column, respectively, defining a position in the array. Since the random row also defines the card value (ace, king, etc.) and the random column defines the suit (spades, hearts, etc.), it is simple to PRINT the card "dealt." The position in the array can be flagged so that any dealt card will not be redealt until the deck has been "shuffled" (by reinitializing the array).

Program 21 simulates dealing a bridge hand (13 cards) and then arranging the hand by suit. This "arranging the hand by suit" introduces a very simple example of a common programming strategy: *sorting* (see statements 930–980). That is, let the program order a given list in either increasing or decreasing value. In the example here, the list is sorted by suit values (1–4) by simply checking each "suit" in the two-dimensional array and PRINTing the value of each card that was "dealt."

However, there are more sophisticated sorting routines. The one shown below sorts 100 or less numbers greater than zero from a one-dimensional array [N(J)] into their increasing numerical order.

```
1000 REM =====BEGIN SORT=====
1010 FOR J = 1 TO 100
1020    D = N(J)
1130       FOR K = J - 1 TO 1 STEP -1
1040          IF N(K) < D THEN 1080
1050          N(K + 1) = N(K)
1080       NEXT K
1070    K = 0
1080    N(K + 1) = D
1090 NEXT J
1100 REM =====END OF SORT=====
1110 FOR I = 1 TO 100
1120    IF N(I) = 0 THEN 1140
1130    PRINT N(I)
1140 NEXT I
```

How could this sorting routine be modified so that it could alphabetize a list of names input into one array, L$( ), as:

LAST NAME(space)FIRST NAME

then print out the sorted (alphabetized) list? (*Note:* An *A* is considered less than a *B*, which is less than a *C* and so on.) Program A732 on the text diskette gives one possible solution.

RUN from disk and refer to the listing and run of Program 21.

PROGRAM 21

```
10 REM PROGRAM 21
20 REM  ===============
30 REM  SIMULATION
40 REM  PROGRAM DEMOS THE USE OF A 2-DIM ARRAY
50 REM  (13 ROWS BY 4 COLUMNS) TO SIMULATE
60 REM  A CARD DECK (13 CARD VALUES BY 4 SUITS).
70 REM  THIRTEEN CARDS (E.G., A BRIDGE HAND)
80 REM  ARE RANDOMLY SELECTED FROM THE 'DECK.'
90 REM  ANOTHER USE OF THE IF-THEN STATEMENT
100 REM IS SHOWN IN COUNTING 'HONOR' POINTS.
130 REM  ===============
140 REM  VARIABLE DICTIONARY
150 REM  ===============
160 REM  C    - A RANDOM CARD 'VALUE' (13-1)
190 REM  C$() - A CARD 'NAME' (ACE, KING, ETC.)
200 REM C(,)  - CARD 'DECK' (A 2-DIM ARRAY)
210 REM  P    - HONOR POINT COUNTER (ACE = 4,
220 REM         KING = 3, QUEEN = 2, JACK = 1)
230 REM  S    - A RANDOM SUIT 'VALUE' (4-1)
260 REM  S$() - SUIT 'NAME' (SPADES, HEARTS, ETC.)
270 REM  ===============
280 DIM C(13,4),C$(13),S$(4)
290 REM  ===CARD NAMES===
300 DATA "ACE","KING","QUEEN","JACK","TEN","NINE"
310 DATA "EIGHT","SEVEN","SIX","FIVE","FOUR","TREY","DEUCE"
330 REM  ===SUIT NAMES===
340 DATA "SPADES","HEARTS","DIAMONDS","CLUBS"
350 REM
360 REM ===STORE THE CARD VALUES (NAMES)===
370 REM
380 FOR I = 1 TO 13 : READ C$(I) : NEXT I
390 REM
400 REM ===STORE THE SUIT VALUES (NAMES)===
410 REM
420 FOR I = 1 TO 4 : READ S$(I) : NEXT I
430 PRINT "♡"
440 PRINT "   A SIMULATED BRIDGE HAND"
450 PRINT
460 REM  ===============
470 REM  INITIALIZE THE ARRAY (SHUFFLE
480 REM  THE CARD DECK)
490 REM  ===============
500 FOR I=1 TO 13
510 FOR J=1 TO 4
520 C(I,J)=0
530 NEXT J
540 NEXT I
550 PRINT"HERE'S HOW THEY WERE DEALT:"
560 PRINT
570 FOR D=1 TO 13
580 REM  ===PICK A CARD VALUE===
580 C = INT(13 * RND(0) + 1)
600 REM  ===PICK A SUIT VALUE===
610 S = INT(4 * RND(0) + 1)
620 REM  ===HAS THIS CARD BEEN DEALT?===
630 IF C(C,S)=1 THEN 590
640 C(C,S)=1
650 REM  ===============
660 REM  HERE'S ANOTHER USE OF IF-THEN STATEMENTS:
670 REM  IF THE EXPRESSION IS TRUE, THEN THE
680 REM  VALUE OF P WILL BE INCREASED ACCORDINGLY.
680 REM  ===============
682 IF C > 4 THEN 770
700 IF C = 1 THEN P = P + 4
710 IF C = 2 THEN P = P + 3
720 IF C = 3 THEN P = P + 2
730 IF C = 4 THEN P = P + 1
740 REM
750 REM ===DISPLAY THE CARD DEALT===
760 REM
770 PRINT TAB(5) C$(C) TAB(11);"OF" TAB(14) S$(S)
780 NEXT D
780 PRINT
800 PRINT"DEPRESS ANY KEY, AND I'LL"
810 PRINT"ARRANGE THE HAND BY SUIT."
820 REM  ===============
830 REM WE'LL ARRANGE THE DEALT HAND BY GOING DOWN
840 REM EACH COLUMN AND SEEING IF A GIVEN CARD HAS
850 REM BEEN PICKED, I.E., DOES C(I,J) = 1?
860 REM IF IT HAS BEEN PICKED, 'I' DETERMINES THE
870 REM CARD FACE VALUE (NAME) AND 'J' THE SUIT.
880 REM NOTE:  IN THE NESTED LOOP BELOW, WE LOOK
880 REM AT THE TABLE BY ***SUIT*** !
800 REM ===============
910 GET Z$ : IF Z$ = "" THEN 810
920 PRINT "♡" : PRINT TAB(5)"ARRANGED BY SUIT:" : PRINT
930 FOR J = 1 TO 4
940 FOR I = 1 TO 13
950 IF C(I,J) = 0 THEN 970
960 PRINT TAB(5) C$(I) TAB(11) "OF" TAB(14) S$(J)
870 NEXT I
```

```
972 PRINT
980 NEXT J
990 PRINT : INPUT "THE 'HONOR' POINTS IN THIS HAND ARE";H
1000 IF H = P THEN PRINT "GO GET 'EM GOREN! THAT'S RIGHT!" :
     GOTO 1020
1010 PRINT "NO, I COUNT" P "HONOR POINTS HERE."
1020 PRINT : INPUT "DEAL ANOTHER HAND (Y OR N)";Z$
1030 PRINT "♡" : IF Z$ ="Y" THEN P = 0 : GOTO 500
1040 FOR I = 1 TO 11 : PRINT : NEXT I
1050 PRINT "MAY YOUR LIFE BE FILLED WITH GRAND SLAMS"
1060 END

RUN
   A SIMULATED BRIDGE HAND

HERE'S HOW THEY WERE DEALT:

    EIGHT  OF  SPADES
    FIVE   OF  CLUBS
    ACE    OF  DIAMONDS
    QUEEN  OF  HEARTS
    SIX    OF  SPADES
    TREY   OF  HEARTS
    SEVEN  OF  CLUBS
    SEVEN  OF  DIAMONDS
    NINE   OF  DIAMONDS
    EIGHT  OF  CLUBS
    DEUCE  OF  SPACES
    ACE    OF  CLUBS
    EIGHT  OF  HEARTS

DEPRESS ANY KEY, AND I'LL
ARRANGE THE HAND BY SUIT.
    ARRANGED BY SUIT:

    EIGHT  OF  SPADES
    SIX    OF  SPADES
    DEUCE  OF  SPADES

    QUEEN  OF  HEARTS
    EIGHT  OF  HEARTS
    TREY   OF  HEARTS

    ACE    OF  DIAMONDS
    NINE   OF  DIAMONDS
    SEVEN  OF  DIAMONDS

    ACE    OF  CLUBS
    EIGHT  OF  CLUBS
```

```
          SEVEN   OF  CLUBS
          FIVE    OF  CLUBS

THE 'HONOR' POINTS IN THIS HAND ARE? 5
NO, I COUNT 10 HONOR POINTS HERE.

DEAL ANOTHER (Y OR N)?N

MAY YOUR LIFE BE FILLED WITH GRAND SLAMS
```

7.3.3 PROGRAM 22: Caloric Intake and Ideal Weight

Diet and the maintenance of proper body weight are popular concerns in our society. It is well known that by careful control of caloric intake and a good exercise program, weight can be lost or gained and then maintained at an "ideal" level.

Program 22 is based upon a model that defines a woman's ideal weight as 90 pounds plus 5 pounds for each inch over (or minus 5 pounds for each inch under) 5 feet in height. A man's ideal weight is defined as 96 pounds plus 6 pounds for each inch over 5 feet in height. This weight, times an "exercise activity factor" of 12 if not active, 15 if moderately active, or 18 if very active, gives an approximate daily calorie count to maintain the ideal weight. Assuming that a pound of fat is equivalent to 3500 calories, a prediction of weekly weight loss or gain can be made.

Program 22 allows manipulation of a limited daily menu and exercise activity factors to examine the effect on weight. However, its use here is primarily that of a simple illustration of basing a simulation on some defined model. Any model (within reason) can be simulated by a computer program. The most important step in developing the program is careful analysis of its design based upon a given model.

Note: For some fun using a loosely based simulation, LOAD "ISLAND",8 and RUN.

RUN from disk and refer to the listing and run of Program 22.

PROGRAM 22

```
10 REM  PROGRAM 22
20 REM  =====================
30 REM  SIMULATION:  THIS PROGRAM PRESENTS A
40 REM  SIMULATED (AND LIMITED!) DAILY MENU
50 REM  FOR SELECTION BY A USER.  BASED UPON
60 REM  THE MENU SELECTED AND THE SEX, HT.,
70 REM  AND ACTIVITY OF THE USER, AN IDEAL
80 REM  WEIGHT AND THE NUMBER OF CALORIES TO MAIN-
90 REM  TAIN THAT WEIGHT IS GIVEN.  FINALLY,
100 REM  A WEIGHT DIFFERENTIAL (LOSS OR GAIN)
```

```
110 REM  ASSUMING CONSISTENT CALORIC INTAKE
120 REM  IS SHOWN, BUT, THE MENU IS LIMITED!
130 REM  ===================
140 REM
150 REM  VARIABLE DICTIONARY
160 REM  ===================
170 REM  A    - EXERCISE ACTIVITY FACTOR
180 REM  B    - BASE WEIGHT
190 REM  C    - CALORIES TO MAINTAIN IDEAL WT.
200 REM  C() - CALORIES PER FOOD PORTION
210 REM  E    - TYPE OF EXERCISE
220 REM  F$()- A LIST OF FOODS FOR EACH MEAL
230 REM  H    - HEIGHT IN INCHES
240 REM  I    - IDEAL WEIGHT
250 REM  M$   - A MEAL (BREAKFAST, LUNCH, DINNER)
254 REM  MC   - CALORIC INTAKE PER MEAL
260 REM  N    - THE NUMBER OF THE FOOD SELECTED
270 REM  P    - POUNDS (LOSS OR GAIN)
280 REM  S    - SEX OF THE USER
290 REM  T    - TOTAL CALORIC INTAKE
300 REM  W    - WT. FACTOR/INCH FROM 5 FT.
310 REM  Z() - FLAG FOR FOOD SELECTED
320 REM  ====================
330 REM
340 DIM C(11),F$(11),Z(11)
350 PRINT "♡";:FOR J=1 TO 9:PRINT:NEXT J
360 PRINT "A SIMULATED DAILY CALORIC INTAKE AND"
370 PRINT:PRINT "  ITS EFFECT ON YOUR IDEAL WEIGHT"
380 FOR J=1 TO 4000:NEXT J
390 PRINT "♡":PRINT"YOU WILL BE PRESENTED A MENU FOR BREAK-"
400 PRINT:PRINT"FAST, LUNCH, AND DINNER. SELECT AS MANY"
410 PRINT:PRINT"ITEMS FROM EACH MENU AS YOU WISH. AFTER"
420 PRINT:PRINT"YOUR DAILY MENU HAS BEEN COMPLETED, YOU"
430 PRINT:PRINT"WILL RECEIVE A SUMMARY OF YOUR CALORIC"
440 PRINT:PRINT"INTAKE AND ITS EFFECT ON YOUR IDEAL WT."
450 PRINT:PRINT"     D E P R E S S   A N Y   K E Y..."
460 GET Z$:IF Z$="" THEN 460
470 REM
480 REM  ===BREAKFAST DATA===
490 REM
500 DATA "BREAKFAST","BACON OR SAUSAGE",200,"CEREAL WITH
    MILK",250
510 DATA "COFFEE (BLACK)",5,"COFFEE (WITH SUGAR)",50
520 DATA "EGGS (2)",100,"MILK",125
530 DATA "ORANGE JUICE",60,"PANCAKES",225
540 DATA "SWEET ROLL",250,"TOAST",75,"WAFFLES",550
550 GOSUB 1290
560 REM
570 REM  ===LUNCH DATA===
580 REM
580 DATA "LUNCH","BEER",125,"BEFORE LUNCH DRINK",115
600 DATA "CHEESEBURGER",310,"COLA",144
610 DATA "COTTAGE CHEESE",110,"CRACKERS",75
620 DATA "FRENCH FRIES",400,"HAMBURGER",260
630 DATA "MILK",125,"TUNA FISH",50
640 DATA "VEGETABLE OR FRUIT SALAD",75
650 GOSUB 1290
660 REM
670 REM  ===DINNER DATA===
880 REM
890 DATA "DINNER","APPLE PIE",300
700 DATA "BAKED POTATO",250,"BEFORE DINNER DRINK",115
710 DATA "BEEF STEAK",560,"BEETS",40
720 DATA "DOZEN RAW OYSTERS",240,"FISH",400
730 DATA "MACARONI",65,"PEAS",115
740 DATA "TOSSED SALAD",75,"T.V. DINNER",500
750 GOSUB 1290
760 GOSUB 1530
770 PRINT "NOW, SOME PERSONAL DATA IS NEEDED..."
780 FOR J=1 TO 3000:NEXT J
790 GOSUB 1530
800 PRINT"ARE YOU:"
810 PRINT"   1. FEMALE"
820 PRINT"   2. MALE"
830 PRINT"ENTER 1 OR 2";
840 INPUT S
850 IF S < 1 OR S > 2 THEN 830
860 IF S=1 THEN B = 90 : W = 5 : GOTO 880
870 B = 96 : W = 6
880 GOSUB 1530
890 PRINT"WHAT IS YOUR HEIGHT IN INCHES";
900 INPUT H
910 IF H < 48 OR H > 84 THEN 890
912 REM
914 REM  ===COMPUTE 'IDEAL' WEIGHT===
916 REM
920 I=((H - 60) * W) + B
930 GOSUB 1530
940 PRINT"DO YOU CONSIDER YOURSELF:"
950 PRINT"   1. SEDENTARY (LITTLE EXERCISE)"
960 PRINT"   2. MODERATELY ACTIVE"
970 PRINT"   3. VERY ACTIVE"
980 PRINT"ENTER 1, 2, OR 3";
990 INPUT E
1000 IF E < 1 OR E > 3 THEN 980
1010 IF E=1 THEN A=12 : GOTO 1040
1020 IF E=2 THEN A=15 : GOTO 1040
1030 IF E=3 THEN A=18
```

```
1032 REM
1034 REM ===COMPUTE CALORIES TO MAINTAIN 'IDEAL' WT.===
1036 REM
1040 C=I * A
1050 PRINT "♡" : PRINT TAB(8) "SUMMARY OF DATA" : PRINT
1060 PRINT "YOUR IDEAL WEIGHT IS" I "POUNDS." : PRINT
1070 PRINT "TO MAINTAIN THAT WEIGHT YOU WILL NEED"
1080 PRINT "ABOUT" C "CALORIES PER DAY." : PRINT
1090 PRINT "YOUR DAILY CALORIC INTAKE BASED UPON"
1100 PRINT "THE LIMITED MENU IS" T "CALORIES." : PRINT
1110 PRINT "   D E P R E S S  A N Y  K E Y..."
1120 GET Z$:IF Z$="" THEN 1120
1130 PRINT "♡" : PRINT : PRINT " D A T A  A N A L Y S I S"
1132 REM
1134 REM ===COMPUTE WEEKLY WT. DIFFERENTIAL===
1136 REM
1140 P=INT((((T - C) * 7) / 3500) * 10)/10
1150 FOR J=1 TO 4 : PRINT : NEXT J
1160 PRINT:PRINT"IF YOU ARE CONSISTENT IN THIS CALORIC"
1170 PRINT:PRINT"INTAKE, YOUR WEIGHT DIFFERENTIAL WILL"
1180 PRINT:PRINT"BE APPROXIMATELY " P "POUND(S)/WEEK."
1190 PRINT:PRINT"   D E P R E S S   A N Y   K E Y..."
1200 GET Z$:IF Z$="" THEN 1200
1210 GOSUB 1530
1220 INPUT "DO YOU WISH ANOTHER ANALYSIS (Y OR N)";Z$
1240 IF Z$ = "Y" THEN T=0 : RESTORE : GOTO 550
1250 GOSUB 1530
1260 PRINT"MAY YOUR BODY BE BEAUTIFUL..."
1270 FOR J=1 TO 4000:NEXT J:PRINT "♡"
1280 END
1282 REM
1284 REM ===SUBROUTINE FOR MENU DISPLAY AND FOOD
     SELECTION===
1286 REM
1290 READ M$
1300 FOR J=1 TO 11:READ F$(J),C(J):NEXT J
1310 PRINT "♡";TAB(14);M$:PRINT
1320 PRINT "   FOOD";TAB(28);"CALORIES":PRINT
1330 FOR J=1 TO 11
1340 IF Z(J)=0 THEN 1390
1350 REM
1360 REM ===REVERSE DISPLAY SELECTED FOOD(S)===
1370 REM
1380 PRINT "R" J TAB(4) F$(J) "-" TAB(30) C(J) : GOTO 1400
1390 PRINT J TAB(4) F$(J)
1400 NEXT J
1410 PRINT " 12 CONTINUE TO NEXT SECTION..."
1412 PRINT : PRINT "MEAL TOTAL:" MC TAB(25) "DAY TOTAL:" T
```

```
1420 PRINT : INPUT " YOUR CHOICE (1 TO 12)";N
1440 IF N < 1 OR N > 12 THEN 1420
1442 REM
1444 REM ===DO WE CONTINUE TO THE NEXT SECTION??===
1446 REM
1450 IF N=12 THEN 1510
1460 REM
1470 REM ===FLAG THE NUMBER OF THE ITEM SELECTED===
1480 REM
1490 Z(N)=1
1492 REM
1494 REM ===ADD CALORIES FOR MEAL AND DAY; DISPLAY MENU
     AGAIN===
1496 REM
1500 MC = MC + C(N) : T = T + C(N) : GOTO 1310
1502 REM
1504 REM ===ZERO SELECTED FOOD LIST BEFORE NEXT MENU===
1506 REM
1510 FOR J=1 TO 11 : Z(J)=0 : NEXT J : MC = 0
1520 RETURN
1522 REM
1524 REM ===SCREEN POSITIONING SUBROUTINE===
1526 REM
1530 PRINT"♡" : FOR J=1 TO 10 : PRINT : NEXT J
1540 RETURN
```

```
RUN

A SIMULATED DAILY CALORIC INTAKE AND
  ITS EFFECT ON YOUR IDEAL WEIGHT
YOU WILL BE PRESENTED A MENU FOR BREAK-
FAST, LUNCH, AND DINNER. SELECT AS MANY
ITEMS FROM EACH MENU AS YOU WISH. AFTER
YOUR DAILY MENU HAS BEEN COMPLETED, YOU
WILL RECEIVE A SUMMARY OF YOUR CALORIC
INTAKE AND ITS EFFECT ON YOUR IDEAL WT.

    D E P R E S S   A N Y   K E Y...

             BREAKFAST

  FOOD                        CALORIES

1. BACON OR SAUSAGE
1. CEREAL WITH MILK
2. COFFEE (BLACK)
4. COFFEE (WITH SUGAR)
5. EGGS (2)
6. MILK
7. ORANGE JUICE
```

```
8. PANCAKES
9. SWEET ROLL
10. TOAST
11. WAFFLES
12. CONTINUE TO NEXT SECTION...

MEAL TOTAL: 0            DAY TOTAL: 0

YOUR CHOICE(S) (1 TO 12)?1

             BREAKFAST

  FOOD                        CALORIES

1. BACON OR SAUSAGE             200
2. CEREAL WITH MILK
3. COFFEE (BLACK)
4. COFFEE (WITH SUGAR)
5. EGGS (2)
6. MILK
7. ORANGE JUICE
8. PANCAKES
9. SWEET ROLL
10. TOAST
11. WAFFLES
12. CONTINUE TO NEXT SECTION...
                  .
                  .
                  .
NOW, SOME PERSONAL DATA IS NEEDED...

ARE YOU:
   1. FEMALE
   2. MALE
ENTER 1 OR 2?2

WHAT IS YOUR HEIGHT IN INCHES?70

DO YOU CONSIDER YOURSELF:
   1. SEDENTARY (LITTLE EXERCISE)
   2. MODERATELY ACTIVE
   3. VERY ACTIVE
ENTER 1, 2, OR 3?2
       SUMMARY OF DATA

YOUR IDEAL WEIGHT IS 156

TO MAINTAIN THAT WEIGHT YOU NEED
2340 CALORIES PER DAY.
```

```
YOUR DAILY CALORIC INTAKE BASED UPON
THE LIMITED MENU IS 2939 CALORIES.

   D E P R E S S   A N Y   K E Y...

     D A T A   A N A L Y S I S

IF YOU ARE CONSISTENT IN THIS CALORIC
INTAKE, YOUR WEIGHT DIFFERENTIAL WILL
BE APPROXIMATELY 1.1 POUNDS/WEEK.

   D E P R E S S   A N Y   K E Y...

DO YOU WISH ANOTHER ANALYSIS (Y OR N)?N

MAY YOUR BODY BE BEAUTIFUL...
```

7.4 Testing

Testing is another application similar to drill-and-practice, with the exception that no assistance is provided. A question is asked, user response is entered, and at some point, the user's performance is indicated.

7.4.1 PROGRAM 23: Name the Seven Dwarfs

Program 23 is a short example of a testing program. This particular program tests the naming of the seven dwarfs of Snow White fame. Names are READ into a one-dimensional array, and then a question loop asks for one of those names. An internal loop searches the list of names for a match. If a match occurs, it is checked for being previously named (flagged). At the conclusion of the program, the complete list is shown and any names in the list not given by the user are starred (*****). Note the use of the one-dimensional array D(n), where n = 1–7, as a flag that prevents double credit for the same name being entered twice. The same flag is also used to "star" those names not entered when the test was taken (see statements 600–690 and 880–950).

Although this program tests on naming dwarfs, the program itself may be used as a general test program. By just changing the DATA and the introductory and closing PRINT statements accordingly, the program could test naming from any chosen list. [Note how an "automatic" DIMensioning can be defined by using a variable name, (*after* a value has been assigned to it), within the DIM statement (see statements 300–330).]

RUN from disk and refer to the listing and run of Program 23.

PROGRAM 23

```
10 REM  PROGRAM 23
20 REM  ==============
30 REM  TESTING
40 REM  PROGRAM DEMOS SIMPLE TESTING EXERCISE
50 REM  IN NAMING.  PROGRAM CHECKS ANY NAME INPUT
60 REM  FIRST FOR ACCURACY AND, IF OK, THEN TO
70 REM  SEE IF NAME HAS BEEN INPUT PREVIOUSLY.
80 REM  ANY NAME NOT ANSWERED IS LISTED AT THE
90 REM  CONCLUSION OF THE PROGRAM.  BY CHANGING
100 REM THE DATA AND THE TITLE AND CLOSING PRINTS
110 REM ACCORDINGLY, THE PROGRAM MAY BE USED AS
130 REM A MODEL FOR TESTING ANY LIST OF NAMES.
140 REM  ===============
150 REM  VARIABLE DICTIONARY
160 REM  ===============
170 REM  D()  - A FLAG FOR THE NUMBER OF THE NAME
180 REM         CORRECTLY ENTERED
190 REM  D$() - THE LIST OF NAMES
210 REM  L    - THE LENGTH OF THE LIST
220 REM  R    - A RANDOM NUMBER (4-1)
230 REM  R$() - POSITIVE FEEDBACK LIST
240 REM  S    - COUNTER FOR THE NUMBER CORRECT
250 REM  ===============
272 REM
274 REM ===STORE THE FEEDBACK===
276 REM
278 FOR I = 1 TO 4 : READ R$(I) : NEXT I
292 REM
294 REM  ===STORE THE LENGTH OF THE LIST===
296 REM
298 READ L
300 REM
310 REM  DIMENSION VARIABLES TO VALUE JUST READ===
320 REM
330 DIM D(L),D$(L)
360 REM
370 REM ===STORE THE LIST OF NAMES===
380 REM
390 FOR I = 1 TO L : READ D$(I) : NEXT I
420 REM
430 REM ===BEGIN THE TEST===
440 REM
450 PRINT "♡"
460 PRINT "SNOW WHITE AND THE 7 DWARFS" : PRINT
480 PRINT "LET'S SEE IF YOU CAN NAME THEM..."
```

```
490 FOR I=1 TO 3000 : NEXT I
500 FOR T = 1 TO L : PRINT "♡"
510 FOR J = 1 TO 10 : PRINT : NEXT J
530 PRINT TAB(6) "NAME NUMBER" T; : INPUT R$
550 REM ===============
560 REM  GO THRU THE LIST TO CHECK FOR A MATCH
570 REM ===============
580 FOR K=1 TO L
590 IF R$ <> D$(K) THEN 750
600 REM ===============
610 REM      AHA! A MATCH FOUND!
620 REM  HAS IT BEEN PREVIOUSLY ENTERED? IF NOT
630 REM  SET D(K) = 1, INCREASE THE SCORE
640 REM  BY 1, AND GIVE A POSITIVE RESPONSE
650 REM ===============
652 IF D(K) = 0 THEN 690
654 REM ===MATCH PREVIOUSLY ENTERED!===
656 A$ = "THAT'S BEEN NAMED ALREADY" : GOSUB 2000 : GOTO 810
660 REM
670 REM ===CORRECT AND NOT PREVIOUSLY NAMED PROCEDURE===
680 REM
690 D(K) = 1 : S = S + 1
700 R = INT(4 * RND(0) + 1) : A$ = R$(R) : GOSUB 2000 : GOTO
    810
750 NEXT K
760 REM ===============
770 REM  IF WE GOT THIS FAR, INPUT NAME DID
780 REM  NOT MATCH ANY NAME IN THE LIST
790 REM ===============
800 A$ = "HMMM...THAT'S NOT ONE" : GOSUB 2000
810 NEXT T
820 PRINT "♡" : PRINT
830 PRINT"DEPRESS ANY KEY FOR THE COMPLETE LIST";
840 GET Z$ : IF Z$ = "" THEN 840
850 PRINT "♡"
860 PRINT
870 PRINT TAB(8) "THE COMPLETE LIST:"
880 FOR I=1 TO L
890 PRINT TAB(12) D$(I);
900 REM ===THOSE CORRECTLY NAMED WERE "FLAGGED" (D(I) =
    1)===
910 IF D(I) = 0 THEN PRINT " *****" : GOTO 950
940 PRINT
950 NEXT I
960 PRINT
970 IF S = L THEN PRINT TAB(6) "*YOU KNEW THEM ALL*" : GOTO
    1030
1010 PRINT TAB(6) "(***** = NAME NOT LISTED!)" : PRINT
1020 REM ===SHOW THE SCORE TO ONE DECIMAL PLACE===
1030 PRINT "THAT'S" INT(S*100/L*10)/10 "PERCENT CORRECT!"
1040 PRINT
1050 PRINT"BYE-BYE FOR NOW...AND WATCH"
1060 PRINT "   OUT FOR THOSE APPLES!"
1070 END
1992 REM
1994 REM ===POSITIONING AND RESPONSE ROUTINE===
1996 REM
2000 PRINT "♡" : FOR J = 1 TO 10 : PRINT : NEXT J
2010 PRINT, A$ " !" : FOR J = 1 to 1000 : NEXT J
2020 RETURN
3000 REM
3010 REM  ===POSITIVE FEEDBACK CHOICES===
3020 REM
3030 DATA "O. K.","G R E A T","S U P E R","V E R Y  G O O D"
3032 REM
3040 REM  ===NO. OF ITEMS IN LIST===
3042 REM
3050 DATA 7
3060 REM
3070 REM  ===LIST OF ITEMS TO BE NAMED===
3080 REM
3090 DATA "BASHFUL","DOC","DOPEY","GRUMPY"
3100 DATA "HAPPY","SLEEPY","SNEEZY"

RUN
SNOW WHITE AND THE 7 DWARFS

LET'S SEE IF YOU CAN NAME THEM...

NAME NUMBER 1?DOC
    G R E A T!

NAME NUMBER 2?HAPPY
    O. K.!
          .
          .
          .

NAME NUMBER 4?GROUCHY
HMMM...THAT'S NOT ONE...

NAME NUMBER 5?GRUMPY
    V E R Y  G O O D!

NAME NUMBER 6?DOC
YOU HAVE GIVEN THAT NAME BEFORE!

NAME NUMBER 7?HAPPY
YOU HAVE GIVEN THAT NAME BEFORE!
```

```
DEPRESS ANY KEY FOR THE COMPLETE LIST
       THE COMPLETE LIST:
           BASHFUL *****
           DOC
           DOPEY *****
           GRUMPY
           HAPPY
           SLEEPY
           SNEEZY *****

    (***** = NAME NOT LISTED!)

THAT'S 57.1 PERCENT CORRECT!

BYE-BYE FOR NOW...AND WATCH
   OUT FOR THOSE APPLES!
```

7.4.2 PROGRAM 24: Multiple-Choice Questions

Program 24 is one example of generating multiple-choice questions. Following any introductory statements, PRINT statements that ask a given question and DATA statements that provide the choices and their appropriate responses may be added to the program. The correct choice by number is assigned to variable A, and then a GOSUB transfers to a subroutine that displays the choices and evaluates the user's input. This sequence of PRINT (the question), DATA (for each choice and its response), A = (number of the correct choice), and GOSUB 5000 may be repeated for an indefinite number of multiple-choice questions in the program. (This program arbitrarily presents only four choices. What would be needed to change the program so that five choices would be displayed?)

RUN from disk and refer to the listing and run of Program 24.

PROGRAM 24

```
10 REM  PROGRAM 24
20 REM  ==============
30 REM  TESTING
40 REM  PROGRAM DEMOS MULTIPLE-CHOICE TESTING
50 REM  USING DATA-READ TECHNIQUES.  QUESTIONS
60 REM  ARE ASKED IN SEQUENCE (I.E., NOT RANDOMLY)
70 REM  ALL QUESTION "SETS" ARE ENTERED IN THE
80 REM  PROGRAM FOLLOWING THE SEQUENCE:
90 REM      1. PRINT STATEMENTS TO ASK THE QUESTION.
100 REM     2. DATA STATEMENTS FOR 4 CHOICES AND
110 REM        A RESPONSE FOR EACH CHOICE.
120 REM     3. SETTING VARIABLE "A" TO THE
```

```
130 REM        NUMBER OF THE CORRECT CHOICE.
140 REM     4. GOSUB 5000
150 REM  MULTIPLE-CHOICE QUESTIONS MAY BE
160 REM  ADDED TO THE PROGRAM IF THIS SEQUENCE
170 REM  IS FOLLOWED.
180 REM  ===============
190 REM  VARIABLE DICTIONARY
200 REM  ===============
210 REM  A    - CORRECT CHOICE ANSWER (1-4)
220 REM  A$() - A GIVEN CHOICE (READ FROM DATA)
230 REM  C    - NUMBER CORRECT COUNTER
240 REM  R    - USER'S ANSWER (VIA INPUT)
250 REM  R$() - A GIVEN RESPONSE (READ FROM DATA)
260 REM  ===============
270 DIM A$(4),R$(4)
280 PRINT "♡" : C = 0
290 REM  ===============
300 REM  ADD INTRODUCTORY STATEMENTS, EXAMPLES,
310 REM  OR WHATEVER HERE (UP TO LINE 500)
320 REM  ===============
330 REM
340 REM
500 REM  ===============
510 REM  PRINT THE QUESTION
520 REM  ===============
530 PRINT "THE STATE FLOWER OF TEXAS IS THE:"
540 REM  ===============
550 REM  AND DATA ELEMENTS FOR EACH
560 REM  CHOICE AND THE RESPONSE FOR THAT CHOICE
570 REM  ===============
580 DATA "BLUE-BONNET","BEAUTIFUL, AREN'T THEY"
590 DATA "ROSE","IT'S BY ANOTHER NAME HERE"
600 DATA "DANDELION","BLOW IT OFF"
610 DATA "MORNING GLORY","IT AIN'T, BUT IT COULD BE"
620 REM  ===============
630 REM  SET A = TO THE CORRECT CHOICE NUMBER
640 REM  ===============
650 A = 1
660 REM  ===============
670 REM  THEN GOSUB 5000 TO PRINT THE CHOICES,
680 REM  GET THE ANSWER, AND THEN RESPOND TO IT.
690 REM  ===============
700 GOSUB 5000
702 REM
710 REM  ===NEXT QUESTION SEQUENCE, ETC.===
712 REM
720 PRINT "AN EXAMPLE OF A COMPUTER OUTPUT"
730 PRINT "DEVICE IS:"
740 DATA "PRINTER","YES, BUT THERE WAS ANOTHER IN THE LIST"
```

```
750 DATA "KEYBOARD","THAT'S AN INPUT DEVICE"
760 DATA "TERMINAL SCREEN","YES, BUT THERE IS A BETTER
    CHOICE"
770 DATA "1 AND 3 ABOVE","O.K...THEY ARE 2 COMMON EXAMPLES"
780 A = 4
790 GOSUB 5000
792 REM
794 REM  ===AND SO ON, AND ON===
796 REM
810 PRINT "WHICH PLANET IS EARTH"
820 PRINT "FROM THE SUN?"
830 DATA "FIRST","THAT'S MERCURY"
840 DATA "SECOND","'TIS VENUS, ABOUT DE MILO FROM THE SUN"
850 DATA "THIRD","RIGHT...YOU'RE TERRA-IFIC"
860 DATA "FOURTH","MAR-CY, THAT'S MARS"
870 A=3
880 GOSUB 5000
890 REM  ===============
900 REM  ROOM TO ADD MANY MORE QUESTION "SETS"
910 REM  FOLLOWING THE SEQUENCE OF:
920 REM  PRINT, DATA, A = , GOSUB
930 REM  ===============
4800 REM
4810 REM
4860 FOR I = 1 TO 10 : PRINT : NEXT I
4870 PRINT "YOU ANSWERED" C "CORRECTLY."
4880 END
4890 REM  ===============
4900 REM  SUBROUTINE TO DISPLAY THE CHOICES, STORE
4910 REM  THE RESPONSE FOR EACH CHOICE, AND
4920 REM  ===============
5000 PRINT
5010 REM  ===============
5020 REM  READ THE DATA FOR THE CHOICE AND
5030 REM  ITS RESPONSE; PRINT THE CHOICE
5040 REM  ===============
5050 FOR I=1 TO 4
5060 READ A$(I),R$(I)
5070 PRINT I TAB(4) A$(I)
5080 PRINT
5090 NEXT I
5100 PRINT
5110 INPUT "YOUR CHOICE (1-4)";R
5120 REM
5130 REM  ===CHECK FOR WITHIN RANGE===
5140 REM
5150 IF R < 1 OR R > 4 THEN 5110
5160 PRINT
5170 REM  ===PRINT THE RESPONSE FOR USER'S CHOICE
```

```
5180 PRINT R$(R) "!"
5190 REM  ===IS IT THE CORRECT CHOICE?===
5200 IF A = R THEN C = C + 1 : GOTO 5320
5210 REM  ===============
5220 REM  IF THE USER'S CHOICE IS NOT CORRECT,
5230 REM  PRINT THE CORRECT CHOICE NUMBER
5240 REM  AND THE CHOICE LISTED
5250 REM  ===============
5260 PRINT
5270 PRINT "A CORRECT CHOICE IS" A ": " A$(A)
5320 PRINT
5330 PRINT"DEPRESS ANY KEY TO CONTINUE..."
5340 GET Z$:IF Z$="" THEN 5340
5350 PRINT "♡" : RETURN
RUN

THE STATE FLOWER OF TEXAS IS THE:
1. BLUE-BONNET
2. ROSE
3. DANDELION
4. MORNING GLORY
YOUR CHOICE (1-4)?3
BLOW IT OFF!
A CORRECT CHOICE IS 1.: BLUE-BONNET
DEPRESS ANY KEY TO CONTINUE...

AN EXAMPLE OF A COMPUTER OUTPUT
DEVICE IS:
1. PRINTER
2. KEYBOARD
3. TERMINAL SCREEN
4. 1. AND 3. ABOVE
YOUR CHOICE (1-4)?1
YES, BUT THERE WAS ANOTHER IN THE LIST!
A CORRECT CHOICE IS 4.: 1. AND 3. ABOVE
DEPRESS ANY KEY TO CONTINUE...

WHICH PLANET IS EARTH
FROM THE SUN?
1. FIRST
2. SECOND
3. THIRD
4. FOURTH
YOUR CHOICE IS (1-4)?4
MAR-CY, THAT'S MARS!
A CORRECT CHOICE IS 3.: THIRD
DEPRESS ANY KEY TO CONTINUE...

YOU ANSWERED 0 CORRECTLY!
```

7.5 Posers and Problems

1. Modify Program 18 to present 8 randomly selected questions from a possible total of 12. Present an introduction to the program and a performance report that gives the number of questions answered correctly on the first, second, and third attempts at its conclusion.
2. Modify Program 23 to test naming from any list of items of your choosing.
3. Modify Program 24 to present five multiple-choice questions of your choosing, each with five possible choices.
4. Identify an area in your particular field of interest in which an instructional computing program could be written for each of the five applications described in Chapters 6 and 7. *Briefly* outline each program by describing in a short paragraph its area, content, and application.

Chapter Eight

You're RIGHT$, I'm LEFT$, She's Gone . . . (But, I'll STRING Along)

Thy knotted and com-
bined strings to part /
And each particular
character to stand on end.
paraphrased from
Shakespeare's
Hamlet, Act 1, Sc. 5

Think About This (for Fun)
Four strangers meet in a room. How many hand-shakes are required for all to be introduced to one another?

Think About This (Seriously)
What characteristics may be identified with educationally valid instructional computing software?

8.1 Objectives

For the successful completion of this chapter you should be able to:

1. Define and give at least one example of "string concatenation" (Section 8.3).
2. Define the purpose of and give at least one example of using the following BASIC functions: ASC, CHR$, LEN, LEFT$, RIGHT$, and MID$ (Sections 8.4.1–8.4.6).
3. Use the Minimum Answer subroutine in at least one program of your choosing (Section 8.5).
4. Use the KEYWORD subroutine in at least one program of your choosing (Section 8.6).

8.2 Some Comments on Strings

As you know by now, many of the instructional computing applications use string input in a program to obtain a user's response. This response is then evaluated, and some form of appropriate feedback is given. To this point, most of this evaluation depended upon an *exact* match between the user's response and some anticipated answer defined in the program. If, however, the user's response contained typographical errors, misspelling, or even something as subtle as a leading or trailing space (blank), and so forth, a match might not occur. This chapter will illustrate the use of certain BASIC functions and programming strategies to better accommodate evaluation of string input. Most of these functions will be incorporated into subroutines—some are simple; some are more complex. Through the use of these subroutines, a user's response may be evaluated in terms of defined "minimum acceptable" or "key phrase" answers. In essence, the subroutines provide a means by which both the response and anticipated answer strings may be examined and compared in much more detail.

8.3 String Concatenation

Strings may be joined together by simple use of the + symbol. This process is called concatenation. As you know, relationships between strings may be compared as in:

```
IF A$ = R$ THEN PRINT "O. K."
```

Other comparisons also may be performed. For example, enter and RUN the following programs.

Program Example:

```
10 PRINT "♡"
20 INPUT "Enter any string of characters";F$
```

```
30 PRINT
40 INPUT "Enter another string of characters";S$
50 PRINT
60 PRINT "The combined strings are " F$ + S$
70 PRINT
80 IF F$ = S$ THEN P$ = " IS THE SAME AS " : GOTO 100
90 P$ = " COMES BEFORE "
100 IF F$ <= S$ THEN PRINT F$ " " P$ " " S$ : GOTO 120
110 PRINT S$ " " P$ " " F$
120 END
```

Program Example:

```
10 PRINT "♡"
20 PRINT "What's the password?"
30 FOR I = 1 TO 3
40    GET C$ : IF C$ = "" THEN 40
50    PW$ = PW$ + C$
60 NEXT I
70 IF PW$ = "C64" THEN PRINT "CORRECT CODE" : GOTO 90
80 PRINT, "INCORRECT CODE!"
90 END
```

8.4 BASIC Functions Related to Strings

8.4.1 Function ASC (STRING$)

Purpose

The ASC function converts the *first* character in STRING$ to its ASCII code (see Appendix B).

Example:

```
10 FOR I = 1 TO 9
20    READ S$
30    PRINT "The ASCII value for " S$ " is " ASC(S$)
40 NEXT I
50 DATA "C","O","M","M","O","D","O","R","E"
```

8.4.2 Function CHR$ (NUMBER)

Purpose

The CHR$ function converts the ASCII code *number* to its character equivalent.

Example:

```
10 FOR C = 65 TO 90
20    PRINT CHR$(C) " ";
30 NEXT C
```

8.4.3 Function LEN (STRING$)

Purpose

The LEN function returns the *number* of characters in STRING$.

Example:

```
10 PRINT "♡" : INPUT "Enter anything at all"; R$
20 L = LEN(R$)
30 PRINT "You typed " L " characters in your response."
```

Enter and RUN the following program.

Program Example:

```
10 PRINT "♡" : INPUT "Enter your full name"; N$
20 PRINT : PRINT "Here is your name underlined:"
30 PRINT N$
40 FOR I = 1 TO LEN(N$)
50    PRINT CHR$(184);
60 NEXT I
```

8.4.4 Function LEFT$(STRING$,NUMBER)

Purpose

The LEFT$ function returns the LEFTMOST characters of "NUMBER" length from STRING$.

Enter and RUN the following examples.

```
10 PRINT "♡" : INPUT "Do you wish to continue"; R$
20 IF LEFT$(R$,1) = "Y" THEN PRINT,"Continuing . . ." : GOTO 40
30 PRINT,"Stopping . . ."
40 END
```

```
10 PRINT "♡" : INPUT "Enter your full name"; N$
20 FOR I = 1 TO LEN(N$)
30    PRINT LEFT$(N$,I)
50 NEXT I
```

8.4.5 Function RIGHT$(STRING$,NUMBER)

Purpose

The RIGHT$ function returns the RIGHTMOST characters of "NUMBER" length from STRING$.

Example:

```
10 PRINT "♡" : INPUT "Enter your name"; N$
20 FOR I = 1 to LEN(N$)
30    PRINT LEFT$(N$,I) TAB(20) RIGHT$(N$,I)
40 NEXT I
```

8.4.6 Function MID$(STRING$,CHARNUM1,CHARNUM2)

Purpose

The MID$ function returns the characters from STRING$ beginning with (character number) CHARNUM1 through CHARNUM2.

Enter and RUN the following example.

```
10 PRINT "CLR/HOME" : INPUT "Enter your full name"; N$
20 L = LEN(N$)
30 PRINT "You entered " L " characters."
40 PRINT "Now we'll 'pull out' some of those characters . . ."
50 INPUT "Enter the beginning character NUMBER you want"; BC
60 INPUT "Now enter the ending character NUMBER you want"; EC
```

```
70 IF BC < 1 OR BC > EC OR EC > L
       THEN PRINT "***OUT OF RANGE***" : GOTO 50
80 PRINT "Those characters are: " MID$(N$,BC,(EC - BC) + 1)
90 PRINT : INPUT "Want to do it again"; Z$
100 IF LEFT$(Z$,1) = "Y" THEN 50
110 END
```

8.5 The Minimum Answer Subroutine

Often in instructional computing, situations arise in a question-answer sequence in which one "key word" or continuous string of certain defined characters will be sufficient for a correct answer. We have seen (ad nauseam) the question:

```
WHAT IS THE CAPITAL OF TEXAS?
```

Of course, the answer anticipated is "AUSTIN" but, until now, answers such as:

```
AUSTIN, TEXAS
```

(or)

```
IT IS THE CITY OF AUSTIN
```

would not match our defined answer. You can imagine the frustration of a student in the following scenario:

```
WHAT IS THE CAPITAL OF TEXAS? AUSTIN
```

(space and RETURN pressed)

```
NO, THE CORRECT NAME IS AUSTIN
```

(And the student thinks, "%$*!@ Stupid computer")

The following routine allows an author to define a "word" of continuous characters as the "minimum acceptable answer" that has been assigned to A$.

```
10000 INPUT R$
10010 L = LEN$(A$)
10020 FOR I = 1 TO LEN(R$ - L + 1)
10030     IF MID$(R$,I,L) = A$
             THEN PRINT "O.K." : RETURN
10040 NEXT I
10050 PRINT "The minimum answer is " A$ : RETURN
```

A LISTing and RUN of a short, sample program is shown below.

Program Example:

```
10 PRINT "♡"
20 PRINT "THIS PROGRAM IS WRITTEN IN A"
30 PRINT "LANGUAGE CALLED -----";
40 A$ = "BASIC" : GOSUB 10000
50 PRINT "THE EPIC POEM 'HIAWATHA' WAS"
60 PRINT "WRITTEN BY ";
70 A$ = "LONGFELLOW" : GOSUB 10000
80 FOR I = 1 TO 12 : PRINT : NEXT I
90 PRINT, "THAT'S ALL . . ."
100 END
10000 INPUT R$
10010 L = LEN(A$)
10020 FOR I = 1 TO (LEN(R$) - L + 1)
10030 IF MID$(R$,I,L) <> A$ THEN 10080
10040 PRINT : PRINT "YES, I'LL ACCEPT YOUR ANSWER," : PRINT
10050 PRINT, R$ : PRINT : PRINT "BECAUSE IT CONTAINS " A$ : GOTO 10090
10060 NEXT I
10070 PRINT : PRINT "YOUR ANSWER DID NOT CONTAIN, " : PRINT
10080 PRINT, A$ : PRINT : PRINT "THE MINIMUM ANSWER I WANTED."
10090 PRINT : PRINT,, "RETURN KEY =>"; : INPUT Z$ : PRINT "♡" : RETURN

RUN
THIS PROGRAM IS WRITTEN IN A
LANGUAGE CALLED -----?ENGLISH
YOUR ANSWER DID NOT CONTAIN
           BASIC
THE MINIMUM ANSWER I WANTED.
                   RETURN KEY =>?
THE EPIC POEM 'HIAWATHA' WAS
WRITTEN BY ?HENRY LONGFELLOW
YES, I'LL ACCEPT
HENRY LONGFELLOW
BECAUSE IT CONTAINS LONGFELLOW
                   RETURN KEY =>?
         THAT'S ALL . . .
```

8.6 The KEYWORD Subroutine

Up to this point, a variety of instructional computing program examples and models have been presented. These programs illustrate some of the major concepts, strategies, and techniques that may be used in program design. However, one additional technique that merits discussion is *keyword matching*.

This technique allows a program author to define a keyword sequence of characters that, if found anywhere in the user's response in the same sequence, will constitute a match between the input and an anticipated answer. For example, assume that an author wanted to ask the following question:

```
YOU HAVE REMOVED YOUR DIRTY SOCKS.
WHAT SHOULD YOU DO WITH THEM NOW?
```

Further assume that the author anticipates the following responses as possible answers to the question:

```
WASH AND DRY THEM
WASH THEM
THROW THEM AWAY
GIVE THEM AWAY
```

With the use of a keyword subroutine, the author can define a match of these anticipated answers as:

```
"WASH*DRY"
"WASH"
"THROW"
"GIVE"
```

Thus, if the user responds with *any* phrase containing the word *throw,* for example, then a match will have been found. The author can then have the program make appropriate responses and transfer execution back to the original question or give the complete answer. If none of the anticipated answers are matched, a response (such as a hint) can be made and execution transferred accordingly.

The program fragment, Keyword, found on the text diskette is one example of a subroutine of this nature. Although many of the program statements are beyond the scope of this text, it is very easy to use the subroutine. However, since Keyword is already written, there are certain conventions that *must* be followed for its successful use:

1. The user's response must be in R$ (i.e., INPUT R$).
2. The defined anticipated answers (keywords to search for in the user's response) must be assigned to A$.
3. A$ may have as many as five keywords, delimited (separated) by an asterisk.
4. If a match occurs between the anticipated answer and the user's response, variable A$ is set to OK. If no match is found, A$ is set to NO. Appropriate response or continuation of keyword checking in the program is then based upon the value of A$.
5. The subroutine begins with statement number 5000.
6. The END statement is number 10000.

Now, referring to the listing of program Socks, note the following sequence for using the subroutine:

1. Program fragment Keyword is loaded from the disk, i.e., the KEY-

WORD subroutine (statements in the range of 5000 through 10000) is loaded into the system's memory.
2. Statement 10 is added to document the program.
3. Statements 20–60 are added to ask the question and assign the user's response to R$.
4. Statement 80 assigns the first anticipated answer (in this case, the correct response) to A$. Two keywords are needed: WASH and DRY. These are delimited by an asterisk. Transfer is then made to the subroutine beginning at statement 5000.
5. Upon return from the subroutine, statement 90 checks the value of A$. If the user's response contains *at least* the keyword WASH followed somewhere by the keyword DRY, a match occurred, and A$ was set to OK by the subroutine. If a match did not occur, A$ was set to NO.
6. If A$ is equal to OK, statement 90 PRINTs the program's response for the keyword defined in statement 80 (the correct answer). Transfer is then to 10000, the END of the program.
7. Statement 110 is executed if no match occurred for the first keyword (A$ = NO). A$ is now redefined as the next keyword to check in the user's response, and transfer is made back to the subroutine. This same sequence of defining a keyword, going to the subroutine, checking the value of A$ upon return, and continuing or responding accordingly is repeated through statement 190.
8. Statements 210–240 are executed if none of the defined keywords are matched, (i.e., either a hint or, after four tries, the correct answer is given).

Following this same sequential strategy, a variety of both *anticipated* correct and incorrect answers may be used in a program. Program Keyword Demo on the text diskette is another example of using the keyword subroutine. Carefully examine its listing. (*Note:* The KEYWORD subroutine portion of the program is *not* included in the listing. It is shown after the RUN of the program.)

Program Example:

```
10 REM ===SOCKS===
20 PRINT "♡"
30 PRINT
40 PRINT "YOU HAVE REMOVED YOUR DIRTY"
50 PRINT "SOCKS.  WHAT SHOULD YOU DO"
60 INPUT "WITH THEM NOW";R$
70 PRINT
80 A$ = "WASH*DRY": GOSUB 5000
90 IF A$ = "OK" THEN PRINT "GOOD!  USE SOME FOOT POWDER TOO!" : GOTO 10000
110 A$ = "WASH": GOSUB 5000
120 IF A$ = "OK" THEN PRINT "DO YOU WEAR WET SOCKS!?" : GOTO 30
140 A$ = "THROW": GOSUB 5000
```

```
150 IF A$ = "OK" THEN PRINT "DON'T TOSS THEM! USE SOME SOAP!" : GOTO 30
180 A$ = "GIVE": GOSUB 5000
190 IF A$ = "OK" THEN PRINT "NO ONE WOULD TAKE THEM, I'LL BET!" : GOTO 30
210 F = F + 1 : IF F < 4 THEN PRINT "THINK OF SOAP AND SUNSHINE!" : GOTO 30
220 PRINT: PRINT
230 PRINT "YOU SHOULD ALWAYS 'WASH' THEN 'DRY'"
240 PRINT TAB(5) "THOSE FILTHY-DIRTY SOCKS!" : GOTO 10000
5000 REM ===KEYWORD SUBROUTINE===

RUN

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?GIVE THEM TO MY MOMMA
NO ONE WOULD TAKE THEM!!!

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?BURY THEM

THINK OF SOAP AND SUNSHINE!

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?WASH THEM FAST
DO YOU WEAR WET SOCKS?

YOU HAVE REMOVED YOUR DIRTY
SOCKS.  WHAT SHOULD YOU DO
WITH THEM NOW?I BETTER THROW THEM AWAY
DON'T TOSS THEM YET...TRY
SOME SOAP AND WATER.

           .
           .
           .

YOU SHOULD ALWAYS 'WASH' THEN 'DRY'
    THOSE FILTHY-DIRTY SOCKS!
```

Program Example:

```
10 REM   PROGRAM NAME: KEYWORD DEMO
20 REM   =========================
30 REM   THIS PROGRAM IS A DEMONSTRATION OF
40 REM   THE "KEYWORD" SUBROUTINE.  UP TO 5
50 REM   KEYWORDS AS DEFINED MUST BE IN THE
60 REM   USER'S RESPONSE FOR A CORRECT ANSWER.
70 REM   OTHER ANTICIPATED ANSWERS ARE CHECKED
80 REM   AND THE PROGRAM RESPONDS ACCORDINGLY
90 REM   IF A MATCH OCCURS.
```

```
100 REM  ========================
110 PRINT "♡" : DIM W$(5)
120 PRINT "A DEMO OF THE KEYWORD SUBROUTINE."
130 PRINT
140 PRINT "WHAT DO WE CALL OUR FIFTY STATES"
150 PRINT "COLLECTIVELY";
155 REM  ===INPUT VARIABLE MUST BE  R$===
160 INPUT R$
164 REM   ==============
165 REM   CHECK FIRST FOR A "CUTE" REPLY,
166 REM   E.G., "NOT THE UNITED STATES OF AMERICA"
167 REM   ==============
168 PRINT
170 A$ = "N*UNITED*STATES"
180 GOSUB 5000
190 REM  ===DID IT MATCH A$ AS DEFINED (A$=OK)?===
210 IF A$ = "OK" THEN PRINT "YOU ARE TRYING TO BE TRICKY!": GOTO 130
222 REM ================
223 REM SET A$ TO NEXT ANTICIPATED ANSWER AND,
224 REM WE'LL DEFINE ANY STRING CONTAINING
225 REM "UNIT STAT AMER" AS CORRECT
230 A$ = "UNIT*STAT*AMER"
240 GOSUB 5000
260 IF A$ = "OK" THEN PRINT"THAT'S IT...VERY GOOD!": GOTO 10000
270 REM   ===SET A$ TO NEXT ANTICIPATED ANSWER===
280 A$ = "UNIT*STAT"
290 GOSUB 5000
310 IF A$ = "OK" THEN PRINT "UNITED STATES OF WHAT???": GOTO 130
320 REM  ===SET A$ TO NEXT ANTICIPATED ANSWER===
330 A$ = "UNITED"
340 GOSUB 5000
360 IF A$ = "OK" THEN PRINT "UNITED WHAT OF WHAT???": GOTO 130
365 REM  ==SET A$ TO NEXT ANTICIPATED ANSWER===
370 A$ = "AMERICA"
380 GOSUB 5000
400 IF A$ = "OK" THEN PRINT "YES, BUT WHAT OF AMERICA???": GOTO 130
405 REM  ==SET A$ TO NEXT ANTICIPATED ANSWER==
410 A$ = "U*S*A"
420 GOSUB 5000
440 IF A$ = "OK" THEN PRINT "HMMMM...THAT SPELLING IS QUESTIONABLE" :GOTO 130
442 REM  ===============
444 REM  NO ANTICIPATED ANSWER WAS MATCHED,
446 REM  SO GIVE THE CORRECT ANSWER.
448 REM  ===============
450 F = F + 1 : PRINT "NO..." 5 - F "ATTEMPT(S) LEFT..." : IF F < 5 THEN 130
460 PRINT "THE ANSWER I WANTED WAS THE"
470 PRINT "UNITED STATES OF AMERICA!"
480 GOTO 10000
5000 REM ===KEYWORD SUBROUTINE===
RUN
A DEMO OF THE KEYWORD SUBROUTINE.

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?THE UNITED STATES
UNITED STATES OF WHAT???

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?AMERICA
YES, BUT WHAT OF AMERICA???

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?UNITED
UNITED WHAT OF WHAT???

WHAT DO WE CALL OUR FIFTY STATES
COLLECTIVELY?THE UNITED STATES OF AMERICA THE HOME OF THE BRAVE AND FREE
THAT'S IT...VERY GOOD!
```

Here is the KEYWORD subroutine:

```
5000 REM ====================
5002 REM PROGRAM NAME: KEYWORD
5004 REM ====================
5006 REM THIS SUBROUTINE READS A
5008 REM USER'S RESPONSE (MUST BE
5010 REM FROM: INPUT R$) AND CHECKS
5012 REM FOR A "KEYWORD" CHARACTER
5014 REM SEQUENCE MATCH AS DEFINED
5016 REM IN A$.  IF A MATCH OCCURS,
5018 REM A$ IS SET TO "OK" OTHERWISE, "NO".
5020 REM NOTE: A$ MAY CONTAIN UP TO 5
5030 REM KEYWORDS DELIMITED BY A *
5040 REM ====================
5042 REM THE SUBROUTINE MAY BE USED WITH ANY
5044 REM PROGRAM BY FIRST LOADING THE
5046 REM "KEYWORD" PROGRAM, THEN ADDING
5048 REM STATEMENTS IN THE SEQUENCE:
5050 REM   PRINT(S) (FOR THE QUESTION)
5052 REM   INPUT R$ (FOR THE USER INPUT)
5054 REM   A$="DEFINED*KEYWORD*ANSWER"
5056 REM   GOSUB 5000
5058 REM   IF A$="OK" THEN PRINT (RESPONSE
5060 REM   FOR MATCH) : GOTO (NEXT QUESTION)
5066 REM   A$="NEXT*KEYWORD"
5068 REM   GOSUB 5000 ETC., ETC...
5070 REM ===================
5170 REM ====================
5180 REM VARIABLE DICTIONARY
5190 REM ====================
5210 REM LA - LENGTH OF ANSWER
5220 REM LR - LENGTH OF RESPONSE
5230 REM LW - LENGTH OF KEYWORD
5240 REM NW - NO. OF KEYWORDS
5250 REM P1,P2 - STRING POINTERS
5260 REM R$ - USER'S RESPONSE
5270 REM W$(5) - ARRAY FOR KEYWORDS
5280 REM ===================
5300 LA = LEN (A$)
5310 LR = LEN (R$)
5320 IF LR < LA GOTO 5620
5330 FOR I = 1 TO 5
5340 FOR J = 1 TO LA
5350 P1 = J
5360 IF MID$ (A$,J,1) = "*" GOTO 5400
5370 NEXT J
5360 P2 = P1
5390 GOTO 5410
5400 P2 = P1 - 1
5410 W$(I) = MID$ (A$,1,P2)
5420 IF P1 < > LA GOTO 5450
5430 NW = I
5440 GOTO 5500
5450 A$ = MID$ (A$,P1 + 1,LA - P1)
5460 LA = LEN (A$)
5470 NEXT I
5500 P1 = 1
5510 FOR I = 1 TO NW
5520 LW = LEN (W$(I))
5530 FOR J = P1 TO LR - LW + 1
5540 P2 = J
5550 IF MID$ (R$,J,LW) = W$(I) GOTO 5580
5580 NEXT J
5570 GOTO 5620
5580 P1 =P2 + LW + 1
5590 NEXT I
5600 A$ = "OK" : RETURN
5620 A$ = "NO" : RETURN
10000 END
```

8.7 Posers and Problems

1. Modify the "Password" program in Section 8.3 to accept "COMMODORE 64" as an alternate password.
2. Write a program of your choosing using the Minimum Answer subroutine.
3. Write a program of your choosing using the KEYWORD subroutine.

CHAPTER NINE

ONE PICTURE IS WORTH TEN THOUSAND WORDS

"What is the use of a book," thought Alice, "without pictures or conversations?"
Lewis Carroll

Think About This (for Fun)
A single English word can be formed from these letters. What is it? Use all the letters: PNLLEEEESSSSS.

Think About This (Seriously)
Is it possible that graphics do not always enhance instructional computing materials?

9.1 Objectives

For the successful completion of this unit, you should be able to:

1. Explain and give an example of how to specify a location on the Commodore's screen (Section 9.2).
2. Describe and give at least one example of how to PRINT the Commodore graphic characters on the screen using a character string (Section 9.3.1).
3. Describe and give at least one example of how to change the character color, the border color, and the background color (Sections 9.3.2 and 9.3.3).
4. Describe and give at least one example of how to PRINT graphic characters using the CHR$ function (Section 9.3.5).
5. Describe and give at least one example of how to PRINT graphic characters of different colors on the screen using the screen memory map and color memory map (Section 9.4).
6. Design, enter, and run a BASIC program of your own design using Commodore color graphics.

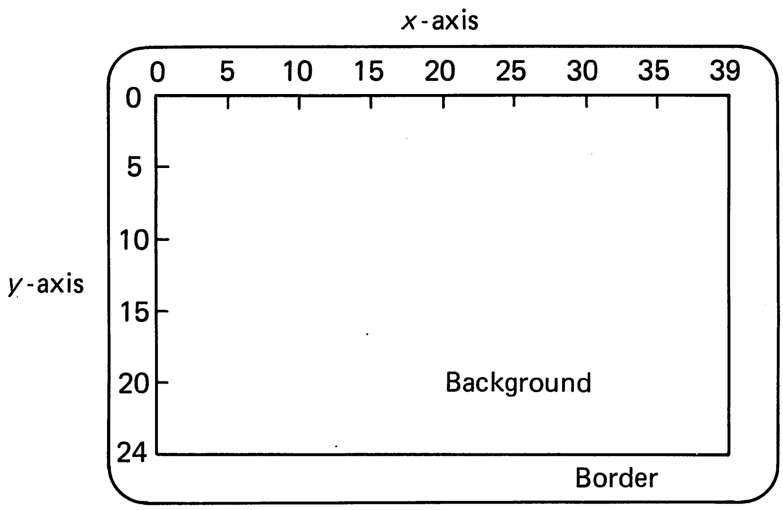
9.2 What Are Graphics?

Throughout history, our civilization's progress has been the result of people's ability to understand complex concepts. Visual tools such as drawings, photographs, films, and video tapes provide the medium for making complex concepts understandable to the masses. With the development of the computer and its ability to analyze vast amounts of data rapidly, the computer's ability to portray visual information (graphics) naturally evolved.

A computer graphic is somewhat like a printed map. Both are two dimensional surfaces with a vertical direction and a horizontal direction. Just as any point on a map can be identified by its horizontal and vertical coordinates (latitude and longitude), any point on a computer's screen can be specified by measuring its vertical and horizontal distance from the upper left corner.

The horizontal distance scale is called by convention the *x*-axis (column variable); the vertical distance scale the *y*-axis (row variable). Figure 9.1 shows the Commodore text screen with the *x*-axis across the top of the screen and the *y*-axis down the left side of the screen. When the position of a character is specified, the distance on the *x*-axis is specified first, followed by the distance on the *y*-axis (the column and row coordinates). For example, 20,12 specifies the location 20 columns to the right in the *x* direction and 12 rows down in the *y*

FIGURE 9.1 Commodore text screen



direction. Similarly, the corners of the Commodore text screen can be specified by 0,0 (upper left); 39,0 (upper right); 0,24 (lower left); and 39,24 (lower right).

9.3 Commodore Text Mode Graphics

The Commodore microcomputers create graphics by using an alternate set of characters on the screen. Color graphics are available in 16 colors but require the Commodore microcomputer to have a color monitor or television connected. Text mode graphics, as their name implies, use character strings and the PRINT statement to create graphics on the normal text screen.

9.3.1 Printing Graphic Characters

There are 62 graphic shapes accessible from the Commodore's keyboard. Each letter key, A through Z, and a few special character keys have their corresponding graphic characters drawn on the front of the keys. The graphic character on the right side of the key can be printed by holding down the SHIFT key while simultaneously pressing the desired shape key. The graphic character on the left side of the key can be printed by holding down the COMMODORE key (the key with the Commodore logo located next to the left SHIFT key) while pressing the desired shape key.

These shapes can be included in a PRINT statement to be printed anywhere on the Commodore's screen like regular text.

Example:

```
PRINT "{SHIFT}S"
PRINT "{COMMODORE}+"
```

In the first example, a heart will be printed on the screen. The second example will cause a checkered box to be printed on the screen. *Note:* In this chapter the use of a key rather than a BASIC statement will be indicated by enclosing the key's name with braces such as {SHIFT} and {COMMODORE}. When you see this type of notation, remember to press the indicated key. Typing {, S, H, I, F, T, and } will *not* create the desired result.

9.3.2 Adding a Little Color

The Commodore has 16 colors that can be used to print both text characters and graphic characters. Normally text and graphics are printed in a shade of light blue. This color can be changed by holding down the CTRL key while simultaneously pressing any number key from 1 through 8. The names of the colors corresponding to the number keys are printed on the front of the keys. Eight additional colors are available by holding down the COMMODORE key while pressing a number key from 1 through 8. The table below indicates the 16 possible colors and their key combinations.

Key combination	**Color**	**Key combination**	**Color**
{CTRL} 1	Black	{COMMODORE} 1	Orange
{CTRL} 2	White	{COMMODORE} 2	Brown
{CTRL} 3	Red	{COMMODORE} 3	Light red
{CTRL} 4	Cyan	{COMMODORE} 4	Gray 1
{CTRL} 5	Purple	{COMMODORE} 5	Gray 2
{CTRL} 6	Green	{COMMODORE} 6	Light green
{CTRL} 7	Blue	{COMMODORE} 7	Light blue
{CTRL} 8	Yellow	{COMMODORE} 8	Gray 3

Just as graphic characters can be incorporated into a PRINT statement, color changes can also be included in a character string in a PRINT statement.

Example:

```
PRINT "{CTRL}3HELLO"
```

The example will print the word HELLO in red letters on the screen.

9.3.3 Changing Border and Background Colors

Background and border colors can also be controlled from BASIC on the Commodore microcomputer. The current background color and border color are stored in two memory locations in the computer. These memory locations must be changed in order to change the border and background colors. To do this from BASIC you must use the POKE statement.

Example:

```
POKE 53280,2
POKE 53281,5
```

In the first example, the number 2 will be POKEd (stored) into the memory location 53280. This is the memory location used to define the border color. In the second example, the number 5 will be POKEd into the memory location 53281. This is the memory location used to define the background color. The following chart lists the border and background color codes used with these two POKE statements:

Border Color:	`POKE 53280,color`
Background Color:	`POKE 53281,color`

```
0 Black          6 Blue           11 Gray 1
1 White          7 Yellow         12 Gray 2
2 Red            8 Orange         13 Light green
3 Cyan           9 Brown          14 Light blue
4 Purple        10 Light red      15 Gray 3
5 Green
```

9.3.4 Somewhere Over the Rainbow

The following color graphics program uses the concepts introduced above. Enter and RUN it:

Program Example:

```
NEW
10 POKE 53280,14
```

Statement 10 changes the border color to light blue, and statement 20 changes the background color to white.

```
20 POKE 53281,1
30 PRINT "♡";
```

Statement 30 clears the screen.

```
40 FOR I=1 TO 22
```

Statement 40 defines a loop to be executed 22 times.

```
50 PRINT TAB(I);
```

Statement 50 TABs over the number of columns equal to the value of the loop counter, I.

```
60 PRINT "{CTRL}5 {SHIFT}* {CTRL}9 {SHIFT}* {CTRL}0";
```

Statement 60 changes the text color to purple, {CTRL}5, PRINTs a triangle on the screen, {SHIFT}, reverses the shape, {CTRL}9, PRINTs the reversed triangle, {SHIFT}*, and reverses the shape back to normal, {CTRL}0.*

```
70 PRINT "{CTRL}7 {SHIFT}* {CTRL}9 {SHIFT}* {CTRL}0";
```
Statements 70–110 repeat the process for each of the remaining colors of the spectrum: blue, green, yellow, orange, and red.
```
80 PRINT "{CTRL}6 {SHIFT}* {CTRL}9 {SHIFT}* {CTRL}0";
90 PRINT "{CTRL}8 {SHIFT}* {CTRL}9 {SHIFT}* {CTRL}0";
100 PRINT "{COMMODORE}1 {SHIFT}* {CTRL}9 {SHIFT}* {CTRL}0";
110 PRINT "{CTRL}3 {SHIFT}* {CTRL}9 {SHIFT}* {CTRL}0";
120 NEXT I
```
Statement 120 completes the FOR-NEXT loop, and statement 130 ENDs the program.
```
130 END
```

Note: To return the screen to its normal border, background, and text colors, hold down the RUN/STOP key while simultaneously pressing the RESTORE key.

9.3.5 String Function CHR$

The CHR$ function returns the character whose code number is supplied as an argument to the function. Every character has what is called its ASCII code (American Standard Code for Information Interchange). These are standard codes for all alphabetic characters and punctuation marks. Commodore microcomputers have their own codes for the special graphic shapes and text colors. See Appendix B.10 for a complete list of the Commodore's ASCII codes.

Example:
```
10 PRINT CHR$(147)
20 FOR X=1 TO 19:PRINT CHR$(29);:NEXT X
30 FOR Y=1 TO 11:PRINT CHR$(17);:NEXT Y
40 PRINT CHR$(144);CHR$(97)
```

The example PRINTs the ace of spades in approximately the center of the screen. CHR$(147) clears the screen, CHR$(29) moves the cursor one column to the right, CHR$(17) moves the cursor one row down, CHR$(144) changes the text color to black, and CHR$(97) prints a spade on the screen.

9.3.6 PROGRAM 25: Random Colored Lines

The text mode graphics described in this chapter provide the basis for adding diagrams, charts, and illustrations to instructional computing materials. Program 25 demonstrates the use of these statements to generate unique art. The program is designed to:

1. Clear the screen.
2. Draw 100 vertical and 100 horizontal lines of varying lengths at random locations on the screen.
3. Draw each line using a random color.

RUN from disk and refer to the listing of Program 25. Since Program 25 is dynamic, it must be RUN to be appreciated. The actions on the screen cannot be sufficiently illustrated by words or pictures in this book.

PROGRAM 25

```
10 REM ===============
```

Statements 10–150 document the program and give a list of the important variables and what they represent.

```
20 REM PROGRAM 25 DESCRIPTION
30 REM ===============
40 REM DEMONSTRATION OF TEXT GRAPHICS.
50 REM THE SCREEN WILL BE COLORED BLUE
60 REM AND 100 RANDOM LINES OF RANDOM
70 REM COLORS WILL BE DRAWN.
80 REM ===============
90 REM VARIABLE DICTIONARY
100 REM ==============
110 REM A - RANDOM STARTING POINT
120 REM B - RANDOM ENDING POINT
130 REM C - RANDOM X OR Y POINT
140 REM X - RANDOM COLOR CODE
150 REM CR$ - COLOR STRING
160 REM ===============
```

The border is set to black, the background to gray, and the screen cleared in statements 160–210.

```
170 REM COLOR IN BACKGROUND
180 REM ===============
190 POKE 53280,0
200 POKE 53281,11
210 PRINT CHR$(147)
220 REM ===============
230 REM LOOP 100 TIMES
240 REM ===============
250 FOR I=1 TO 100
```

Statement 250 defines a loop that is terminated at statement 520. This loop will be executed 100 times.

```
260 REM ===============
270 REM CHOOSE RANDOM COLOR
280 REM ===============
280 GOSUB 2000
```

Statement 290 calls the subroutine at line 2000, which selects a random color.

```
300 REM ===============
310 REM PLOT VERTICAL LINE
320 REM ===============
330 GOSUB 1000
```

Statement 330 calls the subroutine at line 1000, which chooses three random numbers for variables A, B, and C.

Statements 340–380 draw a vertical line from the value of A on the y-axis to the value of B on the y-axis at the point on the x-axis of value C. See if you can figure out the programming logic.

```
340 PRINT CHR$(19);CR$;
350 FOR J=1 TO A:PRINT CHR$(17);:NEXT J
360 FOR J=1 TO B-A+1
370 FOR K=1 TO C+8:PRINT CHR$(29);:NEXT K
380 PRINT CHR$(18);CHR$(32);CHR$(146)
390 NEXT J
400 REM ===============
410 REM ANOTHER RANDOM COLOR
420 REM ===============
430 GOSUB 2000
```

Statement 430 selects another random color.

```
440 REM ===============
450 REM PLOT HORIZONTAL LINE
460 REM ===============
470 GOSUB 1000
```

Statement 470 selects another set of random numbers for variables A, B, and C.

```
480 PRINT CHR$(19);CR$;
```

In statements 480–510, a horizontal line is drawn from the value of A on the x-axis to the value of B on the x-axis at the point on the y-axis of value C.

```
490 FOR J=1 TO C:PRINT CHR$(17);:NEXT J
500 FOR J=1 TO A+8:PRINT CHR$(29);:NEXT J
510 FOR J=1 TO B-A+1:PRINT CHR$(18);CHR$(32);CHR$(146);:NEXT J
520 NEXT I
```

Since statement 520 terminates the loop begun at statement 250, 100 random vertical and 100 random horizontal lines are drawn on the screen.

```
530 END
1000 REM ===============
1010 REM SUBROUTINE TO CHOOSE
1020 REM THREE RANDOM POINTS
1030 REM ===============
1040 LET A=INT(RND(1)*24)
1050 LET B=INT(RND(1)*24)
1060 LET C=INT(RND(1)*24)
1070 IF A=B THEN 1040
1080 IF A<=B THEN RETURN
1090 D=A
1100 A=B
1110 B=D
1120 RETURN
2000 REM ===============
2010 REM RANDOM COLOR SUBROUTINE
2020 REM ===============
2030 LET X=INT(RND(1)*8+1)
2040 ON X GOTO 2050,2060,2070,2080,2090,2100,2110,2120
2050 CR$="0":RETURN
2060 CR$="1":RETURN
```

```
2070 CR$="2":RETURN
2080 CR$="3":RETURN
2090 CR$="4":RETURN
2100 CR$="5":RETURN
2110 CR$="6":RETURN
2120 CR$="7":RETURN
```

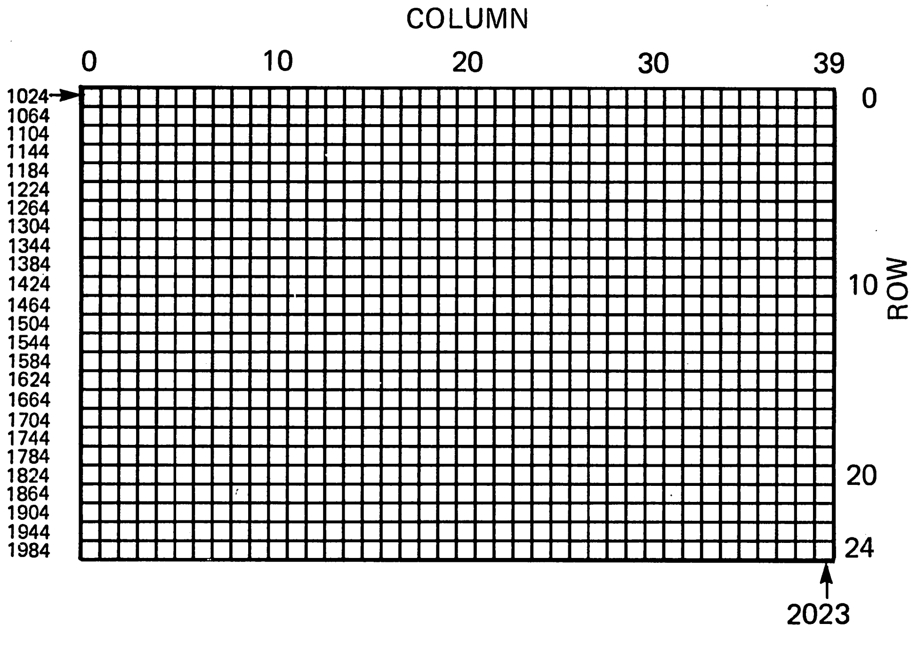
9.4 Commodore Screen Graphics

In addition to being able to print the Commodore graphics characters on the screen with the PRINT statement, the programmer can POKE them directly into the computer's screen memory locations. All microcomputers keep track of which characters are printed where on the screen in an area of the computer's memory called the "screen memory map." On the Commodore, this memory map is available to the programmer through the POKE statement.

9.4.1 The Screen Memory Map

Since the Commodore's screen contains 25 rows of 40 characters, 1000 memory locations are needed for the screen memory map. These locations begin at memory location 1024 and continue through 2023. Figure 9.2 illustrates the Commodore's screen and its associated memory locations.

FIGURE 9.2
Screen memory map



A formula to calculate the location of any point on the screen can be developed. Let X represent the column location from 0 through 39 (left to right), and let Y represent the row location from 0 through 24 (top to bottom). The screen memory map location can be calculated by multiplying Y times 40, adding it to X and the starting memory location, 1024.

Example:

```
POINT = 1024 + X + 40 * Y
```

To place a text or graphic character on the screen, first determine the column and row (X and Y) location and calculate the screen memory location (POINT). Then POKE the screen memory location with the screen display code for the desired character. (*Note*: The screen display codes are not the same as the ASCII codes used in the CHR$ function. See Appendix B.11 for a table of the screen display codes.)

Example:

```
10 X=20
20 Y=12
30 POKE 1024+X+40*Y,90
```

The example will print a diamond in approximately the middle of the screen.

9.4.2 Color Memory Map

The Commodore computer keeps track of the color of each character on the screen just as it keeps track of each character. One thousand memory locations are set aside for this purpose, beginning with memory location 55296 and continuing through 56295. Figure 9.3 illustrates the Commodore's color memory map.

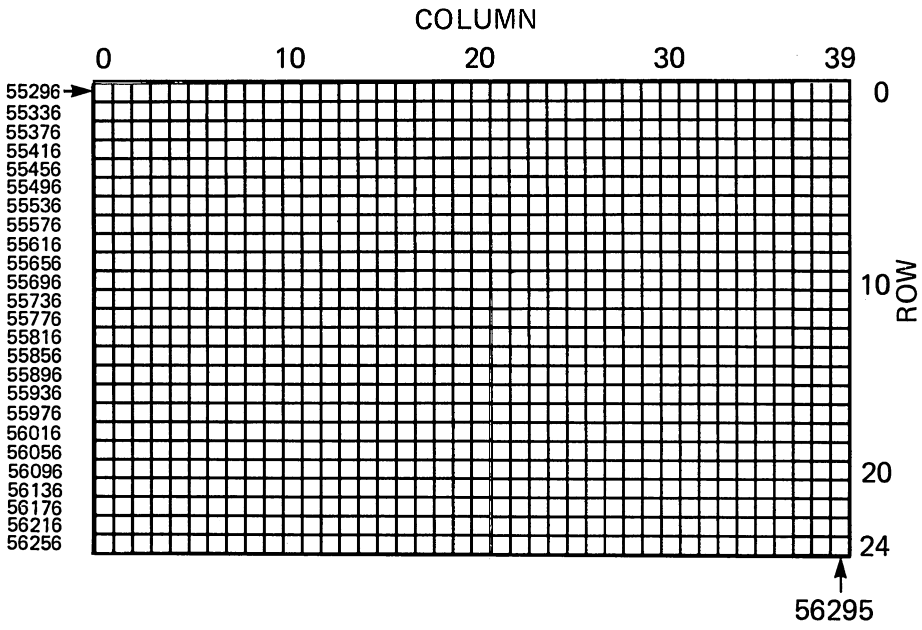
A formula similar to the one used to calculate the screen memory location of a point on the screen can be used to calculate the color memory location. When the color memory location is POKEd with one of the color display codes—the same 0–15 color codes used for the border and background colors (see Section 9.3.3)—the character printed at the corresponding screen location will change colors.

Example:

```
10 X=20
20 Y=12
30 POKE 55296+X+40*Y,2
40 POKE 1024+X+40*Y,90
```

The example prints a red diamond in approximately the middle of the screen.

FIGURE 9.3
Color memory map



9.4.3 A Graphic Example

The following graphics program uses the screen memory map and color memory map. Enter and RUN it:

Program Example:

```
10 POKE 53280,0:POKE 53281,0
```
Statement 10 changes the border and background color to black, and statement 20 clears the screen.
```
20 PRINT CHR$(144)
30 FOR I=1 TO 100
```
Statements 30 and 90 define a loop executed 100 times.
```
40 C=INT(RND(1)*15+1)
```
Statements 40–60 choose a random color (C), a random x-axis position (X), and a random y-axis position (Y).
```
50 X=INT(RND(1)*40)
60 Y=INT(RND(1)*20)
70 POKE 55296+X+40*Y,C
```
Statement 70 sets the color memory map to the selected color, and 80 plots a star at the chosen coordinate.
```
80 POKE 1024+X+40*Y,42
90 NEXT I
100 PRINT CHR$(19)
```
Statements 100–150 print a message.
```
110 FOR R?1 TO 21
120 PRINT CHR$(17);
130 NEXT R
140 PRINT "THE STARS AT NIGHT...ARE BIG AND BRIGHT"
```

```
150 PRINT "       DEEP IN THE HEART OF TEXAS!"
160 GET A$
```

Statements 160 and 170 wait for a key to be pressed, and then 180 ENDs the program.

```
170 IF A$="" THEN 160
180 END
```

9.5 Incorporating Color Graphics into Instructional Computing Materials

When developing instructional computing materials that contain graphics, some special planning is necessary. In addition to the normal designing of the program, the graphic screens used in the program should be sketched or plotted on graph paper. Longer tutorial programs may require a storyboard: a series of sketches of the graphics with the related textual information or questions included.

When designing the program, the graphics are most easily done in subroutines that can be called as needed in the program. The subroutines can be easily tested by typing RUN and the starting line number of the subroutine. (For example, RUN 800 would execute the subroutine beginning at line 800.)

9.5.1 PROGRAM 26: Shape-Recognition Drill

Program 26 is a drill-and-practice program that displays a shape on the screen for the student to identify. Three shapes are used: rectangle, square, and triangle. The program randomly presents ten questions, presents the shape, and keeps track of the student's score. The program elements required in the design are:

1. Instructions to the student.
2. A loop to:
   a. Choose one of the three shapes.
   b. Branch to the appropriate subroutine.
3. Three subroutines (rectangle, square, and triangle) to:
   a. Plot the shape, centered on the screen.
   b. Ask the student to identify the shape.
   c. Input the student's answer.
   d. Display whether the answer is right or wrong.
   e. Tally the correct answers.
4. Display the number of correct answers.
5. End the program.

RUN from disk and refer to the listing of Program 26.

PROGRAM 26

```
10 REM ===============
```

Statements 10–140 document the program.

```
20 REM PROGRAM 26 DESCRIPTION
30 REM ===============
40 REM SHAPE-RECOGNITION DRILL.
50 REM PROGRAM DRAWS A SHAPE ON THE
60 REM SCREEN AND ASKS USER TO IDENTIFY
70 REM IT. SHAPES ARE: RECTANGLE,
80 REM SQUARE, AND TRIANGLE.
90 REM ===============
100 REM VARIABLE DICTIONARY
110 REM ===============
120 REM ANS$ - USER'S RESPONSE
130 REM C - NUMBER CORRECT
140 REM Z - RANDOM SHAPE
150 REM ===============
```

Statements 150–270 provide the student with basic instructions and ask if the student is ready. When the student is ready, the program will continue.

```
160 REM PRINT INTRODUCTION
170 REM ===============
180 PRINT "♡";
190 PRINT
200 PRINT "I AM GOING TO SHOW YOU SOME SHAPES."
210 PRINT
220 PRINT "YOU TELL ME WHAT KIND OF SHAPE IT IS."
230 PRINT
240 PRINT
250 PRINT "ARE YOU READY (Y OR N)";
260 INPUT ANS$
270 IF ANS$<>"Y" THEN 180
280 C=0
```

Statement 280 sets the correct answer counter, C, to zero.

```
290 REM ===============
300 REM ASK 10 QUESTIONS
310 REM ===============
320 FOR I=1 TO 10
```

Statement 320 begins a loop that is terminated at statement 420. This loop is executed ten times.

```
330 PRINT "C";
```

Statement 330 clears the screen.

```
340 FOR J=1 TO 22:PRINT CHR$(17);:NEXT J
```

Statements 340 and 350 PRINT the answer codes at the bottom of the screen.

```
350 PRINT "R=RECTANGLE     S=SQUARE     T=TRIANGLE"
360 REM ===============
370 REM CHOOSE RANDOM SHAPE.
380 REM BRANCH TO SUBROUTINE.
390 REM ===============
400 Z=INT(RND(1)*4+1)
```
A random number from 1 to 3 is chosen at statement 400, and the corresponding subroutine is branched to at statement 410.
```
410 ON Z GOSUB 1000,2000,3000
420 NEXT I
430 PRINT "C";
```
After the loop has been executed ten times, the screen is cleared in statement 430.
```
440 PRINT
450 PRINT "YOU GOT";C;"SHAPES CORRECT!"
```
The number of correct responses is reported to the student in statement 450, and the concluding remarks are made in statement 470. The program ends at statement 480.
```
460 PRINT
470 PRINT "SO LONG FOR NOW."
480 END
1000 REM ======
```
The subroutine beginning at statement 1000 draws a square.
```
1010 REM SQUARE
1020 REM ======
1030 PRINT CHR$(19);
```
Statement 1030 homes the cursor without erasing the screen.
```
1040 PRINT:PRINT
1050 PRINT TAB(10);"_____________"
```
Statements 1050–1090 draw the square.
```
1060 FOR J=1 TO 12
1070 PRINT TAB(10);"|           |"
1080 NEXT J
1090 PRINT TAB(10);"-------------"
1100 PRINT:PRINT "WHICH SHAPE IS IT";
```
Statement 1100 PRINTs, "Which shape is it?"; 1110 inputs the student's answer into the variable ANS$. If the student's answer is "S" for square, execution branches to the subroutine at 4000. If another answer is given, the program branches to the subroutine at 5000.
```
1110 INPUT ANS$
1120 IF ANS$="S" THEN GOSUB 4000
1130 IF ANS$<>"S" THEN GOSUB 5000
1140 RETURN
2000 REM =========
```
The subroutine to draw a rectangle begins at line 2000, and the subroutine to draw a triangle begins at line 3000. The logic for these routines is similar to that for the square. Study them to discover the techniques used.
```
2010 REM RECTANGLE
2020 REM =========
2030 PRINT CHR$(19);
2040 PRINT:PRINT
2050 PRINT TAB( 7);"_________________________"
2060 FOR J=1 TO 10
2070 PRINT TAB(7);"|                       |"
```

```
2080 NEXT J
2090 PRINT TAB( 7);"----------------"
2100 PRINT:PRINT "WHICH SHAPE IS IT";
2110 INPUT ANS$
2120 IF ANS$="R" THEN GOSUB 4000
2130 IF ANS$<>"R" THEN GOSUB 5000
2140 RETURN
3000 REM ========
3010 REM TRIANGLE
3020 REM ========
3030 PRINT CHR$(19);
3040 PRINT:PRINT
3050 PRINT TAB(16);"/\"
3060 FOR J=1 TO 10
3070 PRINT TAB(18-J);"/";SPC(J*2);"\"
3080 NEXT J
3090 PRINT TAB(8);"----------------"
3100 PRINT:PRINT "WHICH SHAPE IS IT";
3110 INPUT ANS$
3120 IF ANS$="T" THEN GOSUB 4000
3130 IF ANS$<>"T" THEN GOSUB 5000
3140 RETURN
4000 REM ==============
4010 REM ANSWER CORRECT
4020 REM ==============
4030 PRINT
4040 PRINT "YOU ARE CORRECT!"
```

The routine beginning at statement 4040 informs the student that the answer was correct, and 4050 adds 1 to the correct answer counter, C.

```
4050 C=C+1
4060 FOR J=1 TO 1000:NEXT J
4070 RETURN
```

Statement 4060 waits for a few seconds, and 4070 RETURNs to the shape subroutine from which it branched.

```
5000 REM ============
```

The routine beginning at statement 5000 informs the student of a wrong answer (statement 5040).

```
5010 REM ANSWER WRONG
5020 REM ============
5030 PRINT
5040 PRINT "SORRY, TRY ANOTHER."
5050 FOR J=1 TO 1000:NEXT J
5060 RETURN
```

Statement 5050 loops 1000 times to slow the pace, and statement 5060 RETURNs to the shape routine from which it branched.

9.6 Some Notes About Using Color

The graphics statements in this chapter can be employed to "add a little color" to instructional computing materials. However, there are both positive and negative factors to be considered when using color:

1. Color can increase attention.
2. Color can increase motivation.
3. Color is less fatiguing to the eye than black and white text.
4. If color is used for highlighting concepts, it must be used consistently throughout the program.
5. Limit the number of colors used at any one time to four.
6. Use highly saturated (bold) colors.
7. Consider color stereotypes. (Stop signs must be red.)
8. The greater the contrast between two colors (i.e., complementary colors), the greater the visual impact.
9. Remember that 10% of all males and 5% of all females are color-blind.
10. *Most important:* If you emphasize everything, nothing on the screen will stand out!

9.7 Posers and Problems

1. Correct any errors in the following statements:

```
10 PRINT "♡"
20 POKE 53000,3
30 POKE 53281,17
30 PRINT "{CTRL}{0}HELLO, CAN YOU COME OUT AND PLAY?"
40 END
```

2. What would result from the execution of the following statements?

```
10 PRINT "♡"
20 FOR N=1 TO 9
30 PRINT CHR$(17);CHR$(29);CHR$(29);
40 NEXT N
50 PRINT CHR$(117);CHR$(113);CHR$(105)
60 FOR N=1 TO 150
70 NEXT N
80 PRINT "♡"
90 FOR N=1 TO 9
100 PRINT CHR$(17);CHR$(29);CHR$(29);
110 NEXT N
120 PRINT CHR$(106);CHR$(113);CHR$(107)
130 FOR N=1 TO 150
```

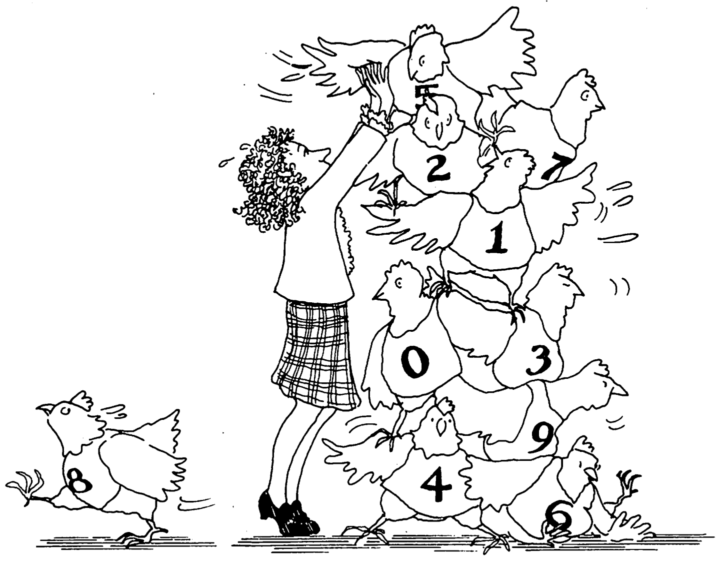
```
140 NEXT N
150 GOTO 10
160 END
```

3. Use the CHR$ function to draw a scene of your choice on the screen (e.g., house, train, space ship, and so on).
4. Write a graphics program that displays a colorful checkered (gingham) pattern on the screen.
5. Write a program that displays 16 bars of different colors on the screen. (This program can be used as a test pattern to adjust the color on the computer's monitor.)
6. Write a program using color graphics to draw rectangles on the screen of random sizes. (*Hint:* Convert Program 25.)

An Introduction to the Design and Development of Instructional Computing Materials

Part Two

Chapter Ten



What Are Your Intentions?

"The trouble with not having a goal is that you can spend your life running up and down the field and never scoring."
Bill Copeland

"It takes less time to do a thing right than to explain why you did it wrong."
H. W. Longfellow

"And what is writ is writ. Would it worthier!"
Lord Byron

"Garbage in, garbage out."
Anon.

"A thing of beauty is a joy forever."
John Keats

Think About This (for Fun)
Using each number only once, arrange the figures 0,1,2,3,4,5,6,7,8,9 so that their sum is 100.

Think About This (Seriously)
Should every student have had an exposure to computers and their uses by the time of graduation from high school?

10.1 Objectives

For the successful completion of this chapter, you should be able to:

1. Identify the steps of a "systems approach" to the design of instructional computing materials (Section 10.3).
2. Identify an area of personal interest within which to apply instructional computing.
3. Outline a rationale, a set of quantitative performance objectives, and a sequence of instruction for a unit of instructional computing materials (Sections 10.3.1–10.3.3).

10.2 Designing Instructional Computing Materials

A working knowledge of BASIC (or any programming language) provides only a very small step toward the actual development of educationally valid instructional computing materials. In fact, such materials have been *designed* by educators with *no* computing experience whatsoever! In these cases, the completed design is given to a computer programmer (who often knows very little about the specific academic area) for translation into an executable computer program. The executable program is tested, refined, and eventually put to use in the classroom. Thus, the key to the development of valid educational materials rests initially with their *design*.

The entire design and development process can be improved if both the author and the programmer have something more than a casual awareness of the other's area of expertise. However, it is not often that the author and programmer are one and the same person, with expertise in both programming and a given academic area. Very few educators have high proficiency in programming techniques and strategies. Likewise, few programmers know the intricacies of learning theory, instructional design, teaching methodology, and so on.

The wide acceptance and use of microcomputers in education is bringing about a gradual change in this, however. More and more, both inservice and preservice teachers are gaining knowledge in computer literacy and instructional computing uses. With this knowledge will come improved materials and improved use of this medium of instructional technology that, literally, is at our fingertips.

Design! It is not too unusual for someone to have the feeling that they have never designed anything! However, if they have ever wanted anything, anything at all, that was eventually obtained through their efforts, they have experienced the design process! This process, then, is really something common to most people, and it has at least one fringe benefit: it makes us think logically and creatively. That is, the procedure—from identification of an objective to its attainment—becomes a series of steps.

Often, this logical procedure is called an *algorithm*, and, in fact, it *is* a logical series of steps that must be followed in designing *any* effective package of instructional materials. This process, however, is amplified greatly in designing and developing interactive instructional computing materials. There are several reasons for this amplification: the primary ones are that instructional computing requires immediate feedback and active user participation. The design of an instructional program—for better or worse—rapidly becomes apparent to a user through its interactive nature.

10.3 The Systems Approach

The design stage of instructional computing materials is one part of a process that is used extensively in the overall development of educational materials. Although this process is known by several names, and the steps may differ slightly among versions, it may be summarized as follows:

1. Statement of the rationale for use
2. Statement of quantitative performance objectives
3. Definition of the instructional sequence
4. Program construction
5. Debugging
6. Pilot testing
7. Revision
8. Use in the classroom
9. Revision
10. Evaluation

These ten steps comprise the process often called a *systems approach* to instructional design. However, since it does involve a logical approach, another name might be, "A Common Sense Approach to Instructional Design."

The first three steps constitute the design stage and will be discussed in this chapter. The following seven steps will be discussed in Chapter 11. Note that, although all of these steps are important, the contents of each are determined solely by the author(s) of the instructional computing materials. In other words, the steps and general procedures for each can be outlined in this book, but the reason for any given instructional computing lesson—what it does and how it does it—can only be determined by its author(s).

10.3.1 The Rationale

Assume that an area of interest has been identified for the design and development of a unit of instructional computing material. Can reasons be stated why this particular area of interest should be taught in the first place? Can reasons be stated why a computer should be used? In other words, the rationale is the answer to *Why?*: *Why* teach this academic concept, and *why* use the computer as an adjunct to the instructional process? If the *why* can not be answered in both instances, the design stage should be terminated and another area of interest identified.

The following examples of rationales are taken directly from instructional computing units developed by various teachers. Note how brief or how thorough such a rationale may be. The first example is very brief:

> The purpose of this learning module (unit) is to enrich the student's personal communication skills, provide a background knowledge for future study in business and economics, and provide a beginning knowledge base of terminology for application in the selected career area. Terminology is essential for communicating in a specialized technological society. This module provides a beginning for building a vocabulary base in business, management, and economics.

The second, slightly longer rationale is very specific:

> Correct association of compound names with molecular formulas is a necessary skill for continuing successfully in a chemistry course. The names and formulas for compounds are used interchangeably throughout most chemical literature. Mastery of chemistry textbook reading material requires the correct identification of compound names and formulas. In the chemistry laboratory, names and formulas are also used interchangeably in labeling containers and in written laboratory procedures. A serious error could result in the laboratory if a student incorrectly identified a compound used in the experiment.
>
> The computer can serve as an effective tool for the student who is learning to identify the names and molecular formulas of compounds because: 1) it allows the student to work at his/her individual pace, 2) it provides immediate feedback to the student after each answer is given, 3) it may randomly generate different questions so that the student has a variety of practice, 4) it scores the student at the end of the drill providing an estimation of progress, and, 5) it may be adapted for use in both drill exercises and testing.

The third example is as specific as the second and is slightly more expansive:

> Preservice educational preparation for nursing in a coronary care unit generally focuses on dysrhythmia recognition. Given various electrocardiographic tracings, the learner is expected to label the patterns by origin

and conduction of impulse, rate, and probable clinical sequela. She/he is rarely provided opportunity to project and evaluate nursing actions based on recognition of the dysrhythmia. Consequently, these decision-making skills are usually learned "on the job" under tutelage of a more experienced nurse practitioner. The trainee's learning depends, then, on numerous variables—the experienced nurse's willingness to teach, clinical situations which "happen" to be present, critical time factors which may or may not permit the trainee opportunity to project appropriate actions before action is required, and numerous other equally uncontrollable factors. Preservice teaching methods can, and should, be developed which facilitate the trainee's acquisition of decision making/judgment skills in environments created deliberately for learning; learning within the setting of a coronary care unit is best reserved for only those abilities that cannot be synthesized in any other environment.

Simulation is one possibly effective preservice teaching technique to facilitate acquisition of decision making/judgment skills. Simulation teaching strategies have been noted to enable the student to: 1) actively participate in learning, 2) integrate theoretical concepts to simulated life situations, 3) desensitize oneself against threatening situations, 4) be presented with identical "hands-on experiences" as those presented fellow learners, 5) experience some of the doubts, competencies, difficulties, and anxieties that would be experienced in actual clinical settings, and, 6) respond in a safe standardized context free of concern about harming the patient or pleasing a tutor.

What are the advantages of using the computer in designing these simulated experiences? First, the selection and sequencing of problems can be randomized independent of instructor or learner choice at the moment—a situation more closely approximating the "randomness" of the actual clinical setting. Second, the learner can be provided with immediate feedback on decisions made. Third, since computers are interactive, the student's response has a measurable effect on the material as it is presented. Fourth, the learner can choose the time for instruction, times when faculty may or may not be available. Fifth, the instructor can reconstruct precisely the sequence in which the student responds to the simulated clinical situation, diagnose errors in approach, and pinpoint reinforcement and help.

In summary, the rationale underlying this unit rests on three premises: 1) A need for preservice acquisition of decision making/judgment skills exists. 2) Simulated experiences can assist in acquisition of these needed skills. 3) Use of the computer enhances the student's independence, assists instructor diagnosis of learning difficulties, and facilitates the process of simulating clinical situations.

10.3.2 Quantitative Performance Objectives

Users will be interacting with your programs. Do they know what is expected of them before, during, and after this interaction? Before a user sits down at a computer terminal, information should be provided that at least outlines the prerequisites for interaction, what the interaction will deal with, and, *specifically*, what constitutes a successful interaction. For what goals should the user strive, and how will it be determined if these goals are attained?

Continuing with our examples from the previous section, a statement of quantitative objectives might be as brief as:

General: Given a basic list of business terms, the student will develop a working knowledge of basic business terms. The student will demonstrate this ability by completing successfully the instructional computing units focusing on terminology mastery.

Specific: Given a set of terminology, the student will complete the instructional computing unit with 90% or better accuracy on a 20-word list.

The second example is succinct and equally brief:

1. The student will be able to state the name of a compound when given its molecular formula with 80% accuracy.
2. The student will be able to state the molecular formula of a compound when given its name with 80% accuracy.

The third example is longer but also quite specific:

Given a cardiac rhythm strip, the student will identify the pattern by site of origin and rhythm with 100% accuracy.

Given a cardiac rhythm strip, the student will identify an appropriate sequence of nursing actions from among the following four alternatives: obtain more data, execute a standing order, call the physician, or continue close observation.

Given a decision to call the physician, the student will indicate the information to be shared, omitting no pertinent data.

Given a decision to obtain more data, the student will ask for data pertinent to formulating a subsequent action-decision.

Given feedback regarding a questionable action decision, the student will re-evaluate the decision and indicate with 100% accuracy if the decision was appropriate.

For a thorough and enlightening description of defining instructional objectives, the reader is referred to the classic text in this field, *Preparing Instructional Objectives* by R. F. Mager (Feron Publications, Palo Alto, CA, 1962).

10.3.3 The Instructional Sequence

This step in design is probably the most difficult for tutorial dialog programs and the least difficult for linear (nonbranching) programs. Obviously, the instructional sequence is in part determined by the type of instructional computing use (problem solving, drill, simulation, and so on) to be applied. This in turn is determined by the rationale, objectives, and interactive tasks defined for the unit. Regardless of the type of use, this step should include, as a minimum, answers to such questions as:

1. Should review material or other information specifically related to the unit be provided prior to actual interaction? If so, what is this information and how will it be provided? What information or examples will be provided by the program itself? Is the program a required or supplemental learning activity?
2. What student control options should be included? Stop at will? Skip problems or sections? Receive answers to questions without an actual attempt at answering? Will help or hints be provided by the program?
3. Will specific content questions be included in the interaction? Are they presented linearly or at random? Will questions be related to model or problem-solving parameters? What model or formula is used? What parameters are needed?
4. What are the anticipated correct answers to requested input? What response(s) will be given? Are ranges possible in the input?
5. What are the anticipated incorrect answers to requested input? What response(s) will be given?
6. What will the program do if neither an anticipated correct nor incorrect answer is matched? Give a hint? Give the answer? How many "misses" will be allowed?
7. Will a "menu" of programs be presented? May the user select at will, or will performance or other criteria determine the sequence?
8. Will branching to review sections be provided for students having difficulty? What will determine that a branch is needed?
9. What constitutes the user's "score"? How is it determined?
10. How will the performance report to the student be presented? Will areas of strength and/or weakness be identified?

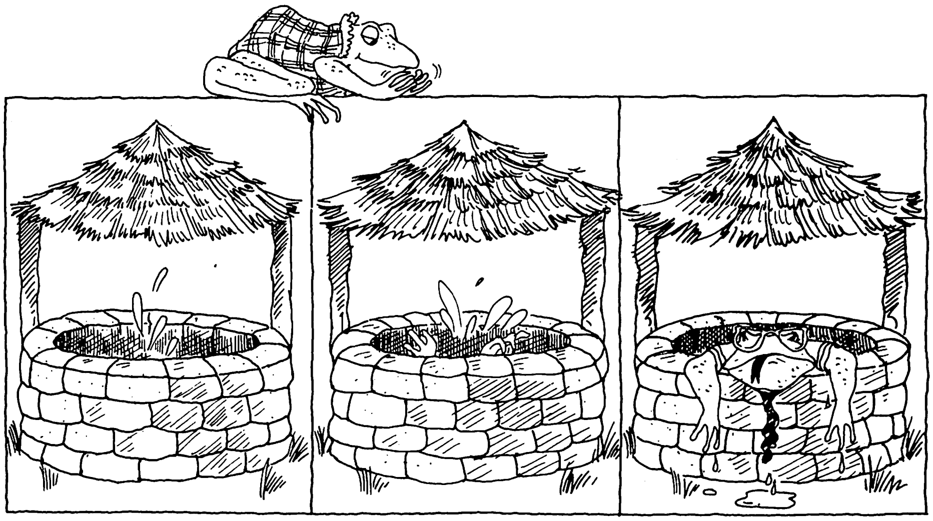
Answers to these—and perhaps many other questions, depending upon the design—must be outlined on paper prior to translation of the defined sequence into a computer programming language.

Most importantly, as the program is being designed, outline each sequential step necessary as it progresses from the opening to the closing "frame" of display. Think through the total sequence of events necessary to impart the concept(s) of the program. Follow this same sequence in a step-by-step manner in the program code. This will ease not only the production of the design but also the translation from paper to actual program. It also provides the added advantage of giving your program good "structure," rather than skipping wantonly from section to section in the program in a "spaghettilike" fashion.

10.4 Posers and Problems

1. Outline on paper the rationale, quantitative objective(s), and sequence of instruction for a short unit of instructional computing in an area of your interest.

Chapter Eleven



Developmental Processes

"The young do not know enough to be prudent and therefore they attempt the impossible—and achieve it, generation after generation."
Pearl S. Buck

"The next-best thing to knowing something is knowing where to find it."
The Ensign

"Them as has, gits."
Unknown

Think About This (for Fun)
A bullfrog is at the bottom of a 30-foot well, trying to escape. Everytime he jumps up 3 feet, he falls back 2. How many jumps will it require for him to escape?

Think About This (Seriously)
Should our society become a computer literate society? If so, how could this be accomplished?

11.1 Objectives

For the successful completion of this chapter, you should be able to:

1. Identify the processes involved in the developmental steps of the systems approach to instructional design (Section 11.2).
2. Identify the 12 guidelines for the design and development of instructional computing materials (Section 11.3).
3. Identify ten guidelines for evaluating software (Section 11.4).
4. Using information discussed in Chapters 1–10, design and develop instructional computing units.

11.2 The Systems Approach (continued)

The design of instructional computing materials constitutes the first three steps of the systems approach. These steps are essentially mental, paper-and-pencil processes. Once the rationale, objectives, and instructional sequence have been defined, the remaining steps of the development process—the coding, debugging, testing, refinement, and use and evaluation of the materials—may be started.

The total process, from rationale to evaluation, for an original set of instructional computing materials may require 50–250 person-hours for each hour of user interaction at a terminal. This would include development of any accompanying materials, such as user and instructor manuals. Of course, if model programs are simply adapted to a user's specific needs, the time required for development is considerably reduced.

11.2.1 Program Construction

Actually, this step is still a mental, paper-and-pencil process for the most part. It primarily involves the translation of the instructional sequence into computer program statements. This is the first of the systematic steps in which some degree of programming expertise is required from either the design author or a programmer. Programming techniques and strategies must be used in transferring the design concepts from paper to executable program code. This step may range from the trivial task of adapting a model program to the extremely involved, time-consuming process of translating an original, detailed design into program code.

11.2.2 Debugging

Once the code has been written, entered, and saved, execution of the program is attempted. Chances are, the program will not run. Problems, commonly called *bugs* in computerese, may be present. These may be anything from simple syntax errors (omitting quotes, misspelling statements, etc.) to technical or conceptual errors (incorrect use of a formula, right answer not accepted, omitting counters, branching at the wrong point, etc.). *Debugging* (extermination of the errors) is done to the point that program execution is satisfactory from the author's viewpoint.

11.2.3 Pilot Testing

Pilot testing of the program is performed next. Generally, this is done with the aid of teaching colleagues and a few volunteer students to test the program on an individual basis. It is recommended that the author literally "look over their shoulders" as they run the program, since it is a rare case in which something unanticipated *does not* occur. These events may be as trivial as the user typing in an anticipated answer, followed by an unanticipated period or space that the program cannot handle. Alternately, a major discussion of the conceptual and/or instructional strategy may be involved. Of course, the main point of pilot testing is *feedback* to the author regarding the design and content of the program.

11.2.4 Revision

It is common for instructional computing materials to be frequently revised. However, the majority of revisions occur after pilot testing. These revisions are usually fairly minor in nature, involving redefining anticipated answers, improving responses, making cosmetic improvement to the display, and so on. However, the revisions could be as major as returning to the design stage for refinement of the program or, in extreme cases, discarding the program. (If the design steps are thought out carefully, this probably will not occur!) Note that the pilot testing and revision steps are cyclic and may be repeated several times prior to the actual classroom use of the program.

11.2.5 Use in the Classroom/Further Revision

Use of instructional computing materials in the classroom is, obviously, directly related to the design of the materials. This use may be supplemental for those students needing review or assistance on a given concept; it may be a required segment of a set of "learning activities"; it may be a prerequisite simulation of a real experiment prior to entering the laboratory; it may be used both as a drill and a testing procedure; and so on.

Regardless of the particular application, it is safe to anticipate minor revision of the materials, if for no other reason than the number of users testing the materials will have increased. Again, it is unlikely that the materials will *ever* get to the point where no additional revisions (however minor) are needed. Thus, use in the classroom and revision are cyclic and may continue as long as the materials are a part of the given instructional process.

11.2.6 Evaluation

Evaluation of instructional computing materials may be divided into two categories. The first is an analysis to determine if users are indeed attaining the defined objectives. This analysis may vary depending upon the design of the materials, but it is often based upon pretest and posttest results. If negative results are indicated, a return to Step 1 of the systems approach may be appropriate.

The second evaluation is that of the *concept* of using instructional computing materials. Did this approach as an instructional medium prove suitable? Analysis of this comes in part from evaluation of the materials in terms of meeting defined objectives. Further evaluation may be based on both user and colleague feedback via attitudinal questionnaires, overall user performance, and, although it lacks quantitative measurement, the author's intuitive feelings.

Note: Research since the late sixties has consistently indicated that the *concept* of the use of supplemental instructional computing materials is educationally valid. In general, the success or failure of any given instructional computing program rests heavily upon the design steps previously discussed. Although it should go without saying, the importance of *thoughtful design* merits emphasis one final time. If, in particular, the rationale, objectives, and instructional sequence are very carefully defined, the chances for successful use of the materials are greatly enhanced. In other words, think it through, folks!

11.3 Guidelines for Design and Development

11.3.1 Consider BASIC

There are a variety of programming languages that may be used in developing instructional computing materials. A variety of "authoring" languages and systems also exist. However, to this point in time, BASIC is the most common language found on microcomputers. Although there are some disadvantages to using BASIC as an instructional computing language (primarily in translating instructional sequence into program code), they are minor when compared to the relative ease of acquiring a working knowledge of the language, its universal nature, and its transportability. These elements, along with the possibility that a working knowledge allows creativity in development, make BASIC worthy of consideration.

11.3.2 Modularize the Units

It is good practice when writing any computer program to keep it as *modular* (concise by topic) as possible. For example, if a given concept includes a series of subconcepts, it is better to have one program for each subconcept, rather than one long program for the total concept. Programs are not only easier to design on this basis but are also easier to debug and revise. Modularization also allows for better "pacing" through an instructional sequence, identification of areas of weakness or strength for a given user, and easier evaluation of a lesson.

11.3.3 Follow a Systems Approach

It is very important that both the author and user of a program know the why, what, how, and effect of using instructional computing materials. Following a systems approach in the design and development of the materials is a means by which this may be accomplished. Above all else, remember that any unit of instruction must be carefully planned and designed "on paper" prior to writing the program code.

11.3.4 State Quantitative Objectives

Although this is one of the steps in the systems approach to instructional design, it merits reiteration. Ensure that users of instructional computing materials know specifically the *extent* and *effect* of a successful interaction with the materials. This means that measurement of the objectives *must* be possible.

11.3.5 Put in Personality

Be kind to the users of your materials. Have a variety of positive reinforcers. Avoid the use of any negative feedback to the user; rather, make your responses to incorrect answers indicate that you are there "in spirit" to assist the user, and then proceed to do so. Include enough humor to solicit a smile or two from the user, but avoid the use of "cute" statements and repetitive responses. Also avoid the use of "fad" responses; they go out of style quickly. Remember that although graphics have appeal, can impart information, can increase motivation, and so on, a "smiling face" that takes 5–10 seconds to be displayed will eventually become boring if given after *every* correct answer.

11.3.6 Consider Gluteal Limits

Another advantage of modularization is that the user will not be sitting at a terminal for lengthy periods. A good rule is to limit the interaction to 30 minutes or less.

11.3.7 Avoid Lengthy Text

Do not make programs "page turners"! It is expensive and boring. One of the key elements in successful instructional computing is that the user be an active learner. If detailed information, figures, tables, and so on, are required, have these available as supplemental materials prior to or during the interaction. In designing the program, consider it a series of "frames" of display, each providing information, giving examples, asking questions, giving feedback, presenting options, and so on. If possible have the user *active* in each frame.

11.3.8 Branch

Another key to success is the individualization that may be provided by branching. If appropriate, the program should have the capability to allow users to review additional material, skip areas if competence is indicated, and/or stop the interaction at will, based upon user need or performance. In any event, never construct a program so that the user is trapped in a routine with no means of escape. Always provide some means by which the user may continue. For example, give the answer after a certain number of incorrect responses or provide other options.

11.3.9 Supplemental Use

For better or worse, the major use of instructional computing is as a supplement or adjunct to traditional instruction. There are *few* courses that are taught by computer alone. Design units that will ease those areas that are routine to the instructional process or that can be best done by instructional computing techniques. Remember, it takes *teachers* to impart personality, lead discussions, and explain abstract concepts.

11.3.10 Document

Your work in the design and development of materials represents much time, effort, and thought. Thus, if possible, have your programs well documented with REMark statements and develop student and teacher guides, where appropriate. This will facilitate not only the local use of your materials, but also their potential use elsewhere.

11.3.11 Review the Literature

Have others done what you are doing? Is their approach different from yours? Are you "reinventing the wheel"? Before you invest the effort required to design and develop materials, you should know what has gone on before. Likewise, if your work is unique and successful, consider publishing a description of what you have done. There are a variety of journals oriented to instructional computing, and other publications available. Others interested in instructional computing should have the opportunity to become aware of your efforts.

11.3.12 Recognize the Capabilities of the Computer

Finally, but perhaps foremost, never forget that, to this point in the realm of instructional computing, computers are an incredibly fast, accurate, and useful *tool.* They can only do what they have been programmed to do. That means that *people* are providing the instructions. Thus, computer programs are only as good or bad in their actions as they have been designed to be by the people who provided the instructions.

Instructional computing materials have been used successfully in problem solving, drill, testing, simulation, and to a lesser degree, tutorial applications. In general, these are applications where speed and accuracy are important in improving the instructional process. That is where we are today.

Where will instructional computing be in the future? More and better of the same? Faster and cheaper computing? Computers in every home and school? Libraries of validated instructional computing materials? Use in practically every academic discipline? It is difficult to accurately predict this future, for the limits are determined by something unpredictable and unlimited: *imagination.*

11.4 Evaluation of Software (Your Own and That of Others)

It is estimated that there are at least 6,000 educational computer programs available in the United States. As you can imagine, some are good, some are poor, and some are in between. In order to begin to review software with a critical eye, some reference point is needed. The following are ten essentials that should be considered as the minimum criteria for software evaluation.

11.4.1 Utilize Unique Capabilities of the Computer

Software should utilize the unique capabilities of the computer. If the use of the computer as an adjunct to the instructional process cannot be justified, then why use it? Ask yourself: What does this software do on the computer that cannot be done in some other manner that is just as effective or efficient?

11.4.2 Pedagogical Use

Software should be pedagogically sound. Is the program based on a learning or instructional theory that is appropriate to the subject matter and cognitive level of the student? Is the program itself pedagogically correct?

11.4.3 Current and Up-to-Date

Software should reflect current and valid curricula. Is the curriculum content of the software consistent with the curriculum content of the classroom? Does it meet the defined goals of the school or district program?

11.4.4 Error-Free

Software should be free of any errors. Are there technical programming errors? Does the program have spelling, grammatical, or content errors? How extensively has the software been validated? Where? How? By whom?

11.4.5 Positive Learning Experience

Software should provide a positive learning experience. Is reinforcement immediate and positive? Many programs make the mistake of rewarding failure with "cute" graphics or sounds. Students may intentionally fail in order to see the results. On the other hand, programs should not have any "humiliating" elements such as sour sound tones for incorrect responses. Software should never contain insulting references to the student's progress, such as "Dummy," or sexual, racial, or religious slurs.

11.4.6 Ease of Use

Software should be easy to use. Are the instructions clear, concise, and consistent? Can the instructions be remembered easily, or can they be recalled if forgotten? Is the operational skill level consistent with the age of the student user? Does the student know the purpose and objectives of the software?

11.4.7 Teacher Modification

Software should allow for teaching modification. Can the teacher select the number or content of questions to be presented? Can the teacher modify the length or instructional sequence of the learning session? Teacher "control" of such options may be an important consideration in some circumstances.

11.4.8 Motivate the Learner

Software should motivate the learner. Its design should be such that frame displays are pleasing, objectives and content are clear, and the learner is an active participant and does not become bored or tired.

11.4.9 Supplemental Materials

Software should provide supplemental written materials. Are there written instructions for the software? Is there a teacher's guide describing the objectives, alternate activities, cross-references, instructional strategies, and evaluation techniques? Are there worksheets, handouts, exercises, tests, and so on?

11.4.10 Backup

Software should provide backup capability. Educational software is costly and can be easily damaged. Are backup copies of the software available? If the software is from a commercial source, can a backup copy be obtained at a reasonable cost? What options does the vendor provide if each student in the class needs a copy of the software? (Remember, we all are responsible for seeing that unauthorized copies of software are not used in the classroom. Not only is this illegal, it also would set bad examples for students.)

Appendix A

The Commodore 64 Computer and How to Use It

A.1 The Commodore 64 Computer

The Commodore 64 microcomputer is one of the most popular computers sold in America. Among the reasons for this popularity is its computing power and economical price. A Commodore owner can begin with an investment of as little as $200 and gradually upgrade the system as his or her interest and budget allow.

The variety of components available for the Commodore 64 make it difficult to describe all the possible combinations. Therefore, this book will limit the discussion to the typical system found in schools:

1. Commodore 64 with 64K of RAM
2. Color monitor
3. Commodore 1541 disk drive
4. Dot matrix printer

A.1.1 Inside the Commodore 64

From the exterior the Commodore 64 appears to be a typewriter with a keyboard but no place to put the paper. Inside the case of the Commodore are the integrated circuits (referred to as "ICs" or "chips") that make it operate. The functional work unit is the microprocessor chip, which is located centrally in the computer. Surrounding the microprocessor are memory chips, peripheral connectors, and other electronics necessary for the operation of the Commodore.

Two types of memory are found in most microcomputers. ROM, read only memory, has programs already stored in it by the manufacturer. These programs are permanent and are never lost even when the power is turned off. RAM, random access memory, in contrast is read and write memory. When the power is turned off, any programs or data stored in RAM are erased.

In the Commodore 64, 20K of ROM (20,480 characters of storage) contain the programs that make the computer operate (the operating system) and the BASIC language interpreter. The latter converts BASIC statements and commands to meaningful codes to which the microprocessor can react.

The Commodore 64 is also outfitted with 64K of RAM (65,536 characters of storage). This memory is used to store a BASIC program, the program's variables, and the text and graphic images on the screen. In the Commodore 1541 disk drive, the Disk Operating System (DOS), which contains the instructions to transfer data and programs between the computer and the drive, resides in 16K of ROM inside the disk drive. An additional 2K of RAM in the 1541 disk drive allows advanced programmers to add commands to the DOS.

Three connectors are provided on the right-hand side and six connectors at the back (see Figure A.1). These connectors (called *buses* by

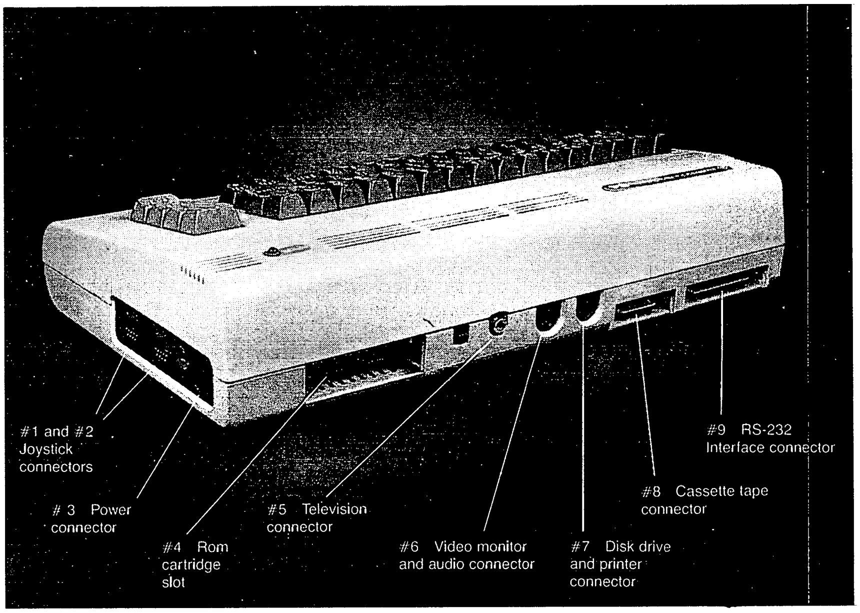

#1 and #2 Joystick connectors
# 3 Power connector
#4 Rom cartridge slot
#5 Television connector
#6 Video monitor and audio connector
#7 Disk drive and printer connector
#8 Cassette tape connector
#9 RS-232 Interface connector

FIGURE A.1
Connectors on the Commodore 64

computer buffs) are used to connect the Commodore with peripheral devices such as joysticks and graphic tablets (#1 and #2 in Figure A.1), the Commodore 64's power supply (#3), ROM expansion cartridges (#4), a television (#5), video monitor and sound equipment (#6), disk drives and printer (#7), cassette tape recorder (#8), and various RS-232 devices like a telephone modem (#9). The disk drive and printer connector (#7) is a serial bus that allows multiple devices to share the bus. In fact you can use up to five disk drives and one printer on this bus. Each 1541 disk drive is "chained" to another using an expansion connector and cable. The printer is connected to the last disk drive at the end of the "chain."

Additional integrated circuits inside the Commodore 64 are used to generate the screen display, decode the keyboard input, and provide support for the RAM, ROM, and microprocessor. As with all electronic appliances, severe damage or shock can result from liquids being spilled inside the Commodore. Appropriate care should be exercised.

A.1.2 The Television (Monitor)

The Commodore 64 will output to any black-and-white or color television. Of course, color graphics cannot be displayed in color on a black-and-white TV. Alternately, either a black-and-white or a color monitor can be used. A monitor will generally produce a sharper picture than a television; however, it is usually more expensive. The TV set is connected to the Commodore through an RF modulator, which converts the Commodore's video signal to a TV signal. The modulator is located inside the Commodore 64 and connects to the TV's antenna leads via a cable plugged into the television connector (#5 in Figure A.1). If a monitor is used, it is connected via cable to the video output connector (#6).

A.1.3 The Commodore 1541 Disk Drive

The 1541 floppy disk drive is the "file cabinet" system of the Commodore 64. It is capable of storing 174,848 characters of information (up to 144 different programs and/or data files) per diskette. The disk drive is connected to the computer via a cable plugged into the serial bus (#7 in Figure A.1). The drive communicates with the Commodore 64 through an interface called a disk controller that is located inside the disk drive cabinet. An expansion connector is provided at the back of the disk drive to allow for additional drives and/or a printer to be connected.

A.1.4 The Dot Matrix Printer

The Commodore 1525 Graphics Printer can be connected to the Commodore 64 through the serial connector (#7) on the computer or through the expansion connector on the 1541 disk drive. A variety of other printers can be connected to the Commodore 64 through an interface plugged into the RS-232 connector (#9 in Figure A.1). The most common and less expensive printers use a pattern of dots to print the characters on the paper, hence the name *dot matrix printer.* The cost of printers ranges from approximately $250 to several thousand dollars, and they are considered by some to be a luxury in the educational setting. However, a printer is essential to the process of developing instructional computing materials.

A.2 How to Use the Commodore 64 with This Text

Accompanying this text is a diskette containing all of the sample programs described in the chapters. This diskette is designed to work on the standard Commodore 64 system with a 1541 disk drive. It is recommended that the reader use this diskette in conjunction with the text in order to study the programs. It is further recommended that a second diskette be used to store the programs you develop from the "Posers and Problems." The following sections will explain how to "power up" the Commodore 64, initialize your own diskette, care for diskettes, use a printer, and deal with trouble if you get into it.

A.2.1 Powering Up

Using the diskette labelled "Instructional Computing Fundamentals for the Commodore 64" power up the system as follows:

1. Turn on the television or monitor and the printer.
2. Turn on the disk drive by pushing upward on the switch located on the back of the drive on your right-hand side. The screen "power" light on the disk drive will come on. For a second, the drive will make whirring sounds while its red light is on.
3. Turn on the Commodore 64 by pushing upward on the switch located on the right-hand side of the computer. The red "power" light will come on on the computer and the disk drive will make whirring sounds again.
4. Open the door on the disk drive and slip the diskette into the slot in the front of the drive with the diskette's label facing upwards. The edge of the diskette with the oval cutout should be toward the back of the drive.

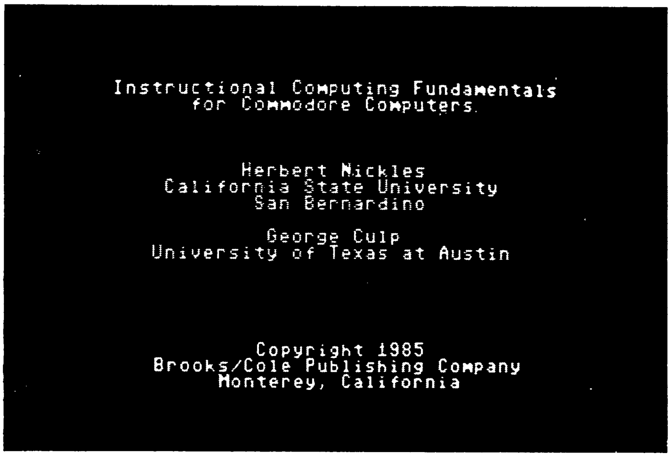
5. Push the diskette gently into the drive until it is entirely inside the drive. Do not force or bend the diskette. Close the disk drive door.
6. Type the following command and press the RETURN key: LOAD "START",8
7. The red light on the disk drive will go on and the drive will make whirring sounds. When the word READY appears on the screen, type RUN and press the RETURN key.
8. In a few seconds, the title of this book and the names of its authors should appear on the screen (see Figure A.2) and then a warning about the use of this diskette. In a few more seconds a menu of programs stored on the diskette will appear (Figure A.3).

To RUN a program from the menu, choose the program, type in its corresponding number and press the RETURN key. When the screen asks whether you wish to LOAD or RUN the program, type the number 2, the option to RUN the program, and press the RETURN key. The computer will automatically LOAD and RUN the program you selected.

To execute a program on the diskette without using the menu, type LOAD followed by the name of the program (in quotes) followed by a comma and the number 8 (the device number for the disk drive) and press the RETURN key. When the computer prints that it is READY, type RUN and press the RETURN key. To see a list of the program's statements, type LIST and hit the RETURN key. For example:

```
LOAD "PROGRAM 1",8 (don't forget the RETURN key)
RUN (RETURN key again)
```

will load Program 1 from the diskette into the computer's memory and execute it.



```
Instructional Computing Fundamentals
     for Commodore Computers.


          Herbert Nickles
   California State University
          San Bernardino

           George Culp
  University of Texas at Austin



          Copyright 1985
  Brooks/Cole Publishing Company
       Monterey, California
```

FIGURE A.2
Booting up the text diskette should result in this title screen

FIGURE A.3
Menu of programs

```
            -------------------------
            W   A   R   N   I   N   G
            -------------------------

THIS IS NOT A DEMONSTRATION DISKETTE!

THE PROGRAMS ARE AN INTERNAL PART OF

AND SOLELY FOR USE IN CONJUNCTION WITH

THE ACCOMPANYING TEXT MATERIAL.

            DEPRESS ANY KEY...

[Clear screen]

* * M E N U   O F   P R O G R A M S * *

EXAMPLE PROGRAMS FROM THE TEXT:

1  6   11  16  21  26
2  7   12  17  22
3  8   13  18  23
4  9   14  19  24
5  10  15  20  25

ANSWERS TO 'POSERS AND PROBLEMS':

27..A354  30..A662  33.A786
28..A422  31..A784
29..A458  32..A785

DEMONSTRATION PROGRAMS FROM THE TEXT:

34..ISLAND        36..SOCKS
35..KEYWORD DEMO  37..MENU

PLEASE ENTER THE NUMBER OF THE PROGRAM
YOU WISH?1
```

A.2.2 Initializing a Blank Diskette

You will want to store the programs you write on a diskette. Although you can store your own programs on the diskette that came with this text, it is best to use another diskette so that you don't accidentally delete a sample program. Obtain a new blank diskette and follow the procedure below:

1. Power up the Commodore 64 as described in steps 1 to 3 in Section A.2.1 above.
2. Insert your blank diskette into the disk drive.
3. Type: OPEN 15,8,15,"NEW:diskname,id" ("Diskname" is any name you wish to give to the disk; "id" is a 2 character identification code of your choice.) Press the RETURN key. The formatting process will begin and the disk drive will make noises for a couple of minutes.
4. When the process is complete, the red light on the disk drive will go out. Remove the diskette and label the outside of the diskette with a pressure sensitive label (use a felt tip pen so that you won't damage the diskette).

It is very important that you have a *blank* diskette in the drive when you follow the above procedure, otherwise you will destroy any programs on the diskette. This procedure formats the diskette so that it can be used with the Commodore 64 only. It cannot be read by any other computer.

A.2.3 Care and Treatment of Diskettes

The programs you store on diskette are valuable. You have an investment in them: either time or money or both. Eliminate troubles by following these simple precautions:

1. Handle a diskette by the jacket (plastic cover) only. Do not allow anything to touch the exposed area of the diskette.
2. Never subject a diskette to a magnetic field as it may erase the diskette. Setting your diskette on top of a TV or printer could cause problems.
3. Keep diskettes flat. Do not fold, bend, or crimp in a three-ring binder.
4. Insert diskettes carefully into the disk drive. Don't use unnecessary force.
5. Store diskettes in their envelopes away from liquids, dirty or greasy surfaces, and dust. In the classroom, chalk dust can cause serious problems with diskettes.
6. Do not expose diskettes to extreme heat or cold. Car dashboards and trunks are diskette killers.

A.2.4 How to Use a Printer

Since several printers may be used with the Commodore 64 computer, the following instructions for using a printer are generalized. Should these instructions not work, refer to your printer or printer interface manual.

1. Locate the on/off switch on the printer and turn it on. (Commodore recommends that the printer be turned on before turning on the computer.)
2. Check for a switch labeled *online/offline* and set it for online.
3. Type OPEN 4,4:CMD 4 and press the RETURN key. Subsequent text which appears on the monitor's screen should also appear on the printer's paper.
4. Printing may be halted by typing PRINT#4:CLOSE 4 and pressing the RETURN key. Locate a switch on the printer labeled *linefeed* or *formfeed.* Use this switch to eject the paper so that the printout can be removed from the printer. If these switches are not present, use the platen knob to "roll" the paper up as you would in a typewriter.

A.2.5 What to Do When All Else Fails

The Commodore 64 is a very simple computer to use. However, even if you have read all the directions, sometimes you may get into a situation that you will need help getting out of. Here are a few suggestions to try:

Halting a runaway. Sometimes when you run a program or take a listing of a program you may desire to stop before it finishes. To do this press the key marked RUN/STOP. It will break (interrupt) the program or listing.

Getting back to normal. When dealing with color graphics, the background and text colors can produce some unexpected results. For example, if the text color is accidentally set to the background color, no text will appear on the screen. To reset the screen to its original text, background, and border colors, press the RUN/STOP key and simultaneously press the RESTORE key. The screen will return to normal, but the program in memory will not be affected.

The last resort. If all attempts to get yourself out of the jam you're in have failed, try turning off the power and following the instructions in Section A.2.1 for powering up the system. Please note that turning off the power can have disastrous results with certain programs. This procedure will definitely erase the program in memory, but will not affect the diskette as long as the red light on the disk drive is not lit when you turn off the power.

If you cannot get the diskette available for this book to load correctly, reread section A.2 to make sure that you are following the directions correctly.

Appendix B

Commodore BASIC Language Summary

This summary defines the most commonly used statements and commands for educators using the Commodore 64 computer. It is not a complete listing of all possible statements, nor does it present detailed descriptions of the action of each statement. The reader who requires such information is referred to the Commodore 64 Programmer's Reference Guide available from Commodore dealers.

The assumption of this guide is that the statements and commands as described are intended to be used on a Commodore 64 computer with a 1541 disk drive. This configuration is very common for educational users. If the reader's system is not configured in this fashion, some of the following statements and commands will function differently than documented.

In the following summary, the general format for each statement or command will be presented with an example, or examples when necessary for clarity, and a description of the action initiated. The conventions and abbreviations used are as follows:

<...>	Required element
{...}	Optional element
cond	Any logical condition
dimension(s)	The maximum dimension(s) of an array
expr	Any numeric constant, variable or expression

file	Any legal filename
key	Any key on the computer's keyboard
line number	Any legal line number from 0 to 63999
message	Any combination of characters
statement	Any legal Commodore BASIC statement
string	Any string constant, variable, or expression
variable or var	Any legal variable described in Section 3

B.1 BASIC Statements

DATA

line number DATA <list of variables>

```
210 DATA 4.3,"A to Z",10
```

Provides a program with data that can be stored into variables using the READ statement. In the example, 4.3 is a real number, "A to Z" is a string, and 10 is an integer. (See READ below.)

DIM

line number DIM <variable(dimension(s))>

```
10 DIM A(23),B(3,4),C$(4),D$(12,30)
```

Defines a variable that is capable of storing a list (single dimension) or a table (double dimension) of a specified length. In the example, A is a numeric variable with 23 possible entries. D$ is a string variable with a maximum of 12 rows and 30 columns.

END

line number END

```
32767 END
```

Terminates the execution of a program

FOR

line number FOR <var> = <expr> TO <expr> {STEP <expr>}

```
45 FOR I=2 TO 10 STEP 2
```

Creates a "loop" that will execute all of the statements between a FOR and a NEXT statement a specified number of times. In the example, the loop would be executed for the values of I from 2 to 10 by 2's (i.e. 2, 4, 6, 8, and 10). (See NEXT below.)

GET

line number GET <variable>

```
70 GET X$
```

Inputs a single character from the keyboard without the character being printed on the screen. Does not require the RETURN key to be pressed. In the example, the input character is stored in the variable X$.

GOSUB

line number GOSUB <line number>

```
220 GOSUB10000
```

Unconditionally branches program execution to a subroutine at the indicated line number. When a RETURN statement is encountered in the subroutine, execution is returned to the statement immediately following the GOSUB. The example will cause the program to branch to the subroutine beginning at line 10000. (See RETURN below.)

GOTO

line number GOTO <line number>

```
670 GOTO 10
```

Causes the execution of the program to branch to the indicated line number. In the example, program execution will branch from line 670 to line 10.

IF-THEN

line number IF <cond> THEN <statement(s)>
line number IF <cond> THEN <line number>
line number IF <cond> GOTO <line number>

```
55 IF A$ = "Y" THEN PRINT "CORRECT"
75 IF X < Z THEN 300
90 IF C <= 578 GOTO 4000
```

Causes the program to execute the indicated statement(s) or branch to a line number if a specified condition is true. If the condition is false, the statement or branch is not executed, and the program continues with the execution of the next numbered statement following the IF-THEN. In the first example, the word CORRECT will be printed if A$ has the string value "Y". The second example will cause a branch to line 300 if the value stored in X is less than the value in Z. The third example will cause a branch to line 4000 if the value of C is less than or equal to 578.

INPUT

line number INPUT {string;} <list of variables>

```
240 INPUT "WHAT IS YOUR NAME ";NAME$
800 INPUT A,B,C
```

Inputs data from the keyboard to be stored into respective variables listed. Optionally, INPUT can print a string on the screen before waiting for input. The RETURN key must be pressed after the user has entered data. In the first example, the string WHAT IS YOUR NAME will be printed on the screen followed by a ? and the cursor. The string the user enters will be stored in NAME$. The second example will input from the keyboard three numeric values separated by commas and store them into A, B, and C respectively.

LET line number LET <variable> = <expr>
line number <variable> = <expr>

```
110 LET C = 100
120 P$ = "GREAT!"
130 A = 1/2 * B + H
```

Assigns the value of <expr> to <variable>. The word LET is optional. In the examples, the value 100 is stored in the variable C, the string GREAT! is stored in the variable P$, and variable A will have the value of one-half B plus H.

NEXT line number NEXT <variable>

```
80 NEXT I
```

Terminates a loop begun by a FOR statement. The <variable> must be the same used in the corresponding FOR statement. In the example, line 80 will terminate the preceding statement: 45 FOR I=2 to 10 STEP 2. (See FOR above.)

ON-GOSUB line number ON <expr> GOSUB <list of line numbers>

```
30 ON X GOSUB 10000,15000
```

Branches to the subroutine at the line numbers indicated based on the arithmetic value of an expression. In the example, the program will branch to the subroutine at line 10000 if X is 1 and to the subroutine at 15000 if X is 2. If X is 0 or greater than 2, the statement immediately following the ON-GOSUB will be executed. If X is less than 0, an "Illegal Quantity" error will occur.

ON-GOTO line number ON <expr> GOTO <list of line numbers>

```
40 ON X-Y GOTO 500,600,700
```

Branches to the line numbers indicated based on the arithmetic value of an expression. In the example, the program will branch to line 500 if X-Y has the value 1, line 600 if X-Y has the value 2, and line 700 if X-Y has the value 3. If X-Y is 0 or greater than 3 then the statement immediately following the ON-GOTO will be executed. If X-Y is less than 0, an "Illegal Quantity" error will occur.

PEEK line number <variable> = PEEK (<expr>)

```
50 X = PEEK(53280)
```

Returns an integer value from 0 to 255 that is stored in the memory location defined by the decimal expression. In the example, the value of X will be set to the integer value of the memory location 53280 which is used to store the color of the border on the screen.

POKE line number POKE <expr1>,<expr2>

```
60 POKE 54296,15
```

Allows the user to change any RAM memory location. <expr1> is the decimal value of the memory location. <expr2> is the value to be placed in memory. In the example, the value 15 will be placed into memory location 54296, which will set the speaker volume to its maximum.

PRINT line number PRINT <list of variables>

```
890 PRINT "YOU GOT";N;"QUESTIONS CORRECT."
```

Causes the computer to advance the cursor to the next line on the screen and print the values of the specified variables or strings. In the example, if N had the value 9, then YOU GOT 9 QUESTIONS CORRECT . would appear on the screen. See Section 2, "Text-Formatting Statements," for more information.

READ line number READ <list of variables>

```
465 READ X,Y,Z
```

Used in conjunction with the DATA statement to store data into variables within a program. When a READ statement is executed, the program will set the variables listed to the next successive values in the program's DATA statements. The example will take the next three values from the DATA statements and store them in X, Y, and Z respectively. (See DATA above.)

REM line number REM <message>

```
10 REM  PROGRAM BY IMA TEACHER
```

Inserts a REMark into the program. The message only appears when the program is listed, not when it is run.

RESTORE line number RESTORE

```
360 RESTORE
```

Returns the DATA list pointer to the first value of the first DATA statement allowing for the DATA to be reread.

RETURN line number RETURN

```
10450 RETURN
```

Terminates a subroutine and returns execution to the next numbered statement following the GOSUB that called the subroutine. (See GOSUB above.)

B.2. Text-Formatting Statements

COMMA (,) line number PRINT <var>,<var>

```
370 PRINT QUANTITY,PRICE,TOTAL
```

Used in a PRINT statement to space data into 10 column fields. In the example, the value of the variable QUANTITY will be printed starting in column 1, the value of the variable PRICE will be printed starting in column 11, and the value of the variable TOTAL will be printed starting in column 21.

CLEAR SCREEN (CLR/HOME KEY)

line number PRINT "♡"

```
10 PRINT "♡"
```

The CLR/HOME key, when used with the SHIFT key, clears the text screen and returns the cursor to the "home" position in the upper left-hand corner. To execute this function, press the SHIFT key while simultaneously pressing the CLR/HOME key. When the keys are included inside parentheses in a PRINT statement, the screen will be cleared when the PRINT statement is executed.

POS line number <var> = POS <expr>

```
730 X=POS(0)
```

POS is a text function that returns the current horizontal cursor position from the left margin (0) to the right margin (39). Although <expr> is required, the expression has no effect on the results. In the example, X will be set to the current horizontal cursor position.

REVERSE CHARACTERS (RVS ON KEY)

line number PRINT "<CTRL><RVS ON><string>"

```
345 PRINT "<CTRL><RVS ON>CORRECT"
```

Sets the text printing mode to reverse video. Text printed following <CTRL><RVS ON> will be printed in reverse video. <RETURN> or <CTRL><RVS OFF> will return to normal video. In the example, CORRECT will appear in reverse video on the screen.

SEMICOLON (;)

line number PRINT <string>;<var>

```
840 PRINT "YOU GOT";N;"CORRECT."
```

Used in a PRINT statement to position the cursor immediately after the string or variable preceding the semicolon. If N = 10 in the example, the printed line would read:

YOU GOT 10 CORRECT.

SPC line number PRINT <var>;SPC(<expr>);<var>

```
480 PRINT A;SPC(10);B
```

Used in a PRINT statement to insert a specified number of spaces between two variables when preceded and followed by semicolons. In the example, the value of A will be printed, followed by 10 spaces and then the value of B will be printed.

TAB line number PRINT TAB (<expr>);<var>

```
80 PRINT TAB(15);R
```

Used in a PRINT statement to move the cursor to the specified column where 0 is the left margin. TAB can only move the cursor to the right. In the example, the value of R will be printed starting in column 15.

B.3 Summary of Variable Types

INTEGER Variable name: single letter optionally followed by a single letter or digit followed by a % character.

Range: `-32768 to +32767`

Examples: `1%, B2%, GH%`

REAL Variable name: single letter optionally followed by a single letter or digit.

Range: `-1.7 E+38 to +1.7 E+38`

Examples: `S, R5, DE`

STRING Variable name: single letter optionally followed by a single letter or digit followed by a $ character.

Range: `0 to 255 characters`

Examples: `F$, K9$, XY$`

Note that variable names may be longer than two characters, but only the first two characters are significant. Consequently VIC and VICTORY are the same real variable VI.

B.4 Summary of Operators

ARITHMETIC

+	Addition	*	
/	Division	–	Subtraction or negation
↑	Exponentiation (raise to a power)		

LOGICAL

AND	Logical product
NOT	Logical negation
OR	Logical sum

RELATIONAL

=	Equals
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

STRING

+	Concatenation

B.5. Mathematical Functions

ABS line number <var> = ABS(<expr>)

```
100 X=ABS(-6.75)
```

Returns the absolute value of <expr>. In the example, X = 6.75.

ATN line number <var> = ATN(<expr>)

```
100 X=ATN(1)
```

Returns the arctangent of <expr> in radians. In the example, X = .785398163.

COS line number <var> = COS(<expr>)

```
100 X=COS(1)
```

Returns the cosine of <expr>. <expr> must be in radians. In the example, X = .540302306.

EXP line number <var> = EXP(<expr>)

```
100 X=EXP(1)
```

Returns the value e ↑ <expr> where e = 2.718289. In the example, X = 2.71828183.

INT line number <var> = INT(<expr>)

```
100 X=INT(4.53)
```

Returns the greatest integer in <expr> which is less than or equal to <expr>. In the example, X = 4.

LOG line number <var> = LOG(<expr>)

```
100 X=LOG(2)
```

Returns the natural logarithm of <expr>. In the example, X = .693147181.

RND line number <var> = RND(<expr>)

```
100 X=RND(0)
```

Returns a random number greater than or equal to 0 and less than 1. If <expr> is 0, RND returns a random number generated from the internal clock. If <expr> is a positive number, the same set of random numbers is generated each time the program is run.

SGN line number <var> = SGN(<expr>)

```
100 X=SGN(-217.456)
```

Returns the sign of <expr>: +1 if positive, 0 if zero, and −1 if negative. In the example X = −1.

SIN line number <var> = SIN(<expr>)

```
100 X=SIN(1)
```

Returns the sine of <expr>. <expr> must be in radians. In the example, X = .841470985.

SQR line number <var> = SQR(<expr>)

```
100 X=SQR(16)
```

Returns the square root of <expr>. In the example, X = 4.

TAN line number <var> = TAN(<expr>)

```
100 X=TAN(1)
```

Returns the tangent of <expr>. <expr> must be in radians. In the example, X = 1.55740772.

B.6 String Functions

ASC	line number <var> = ASC(<string>)

```
100 X=ASC ("COMMODORE")
```

Returns the ASCII code for the first character in the string specified. In the example, X = 67.

CHR$	line number <string> = CHR$(<expr>)

```
100 X$=CHR$(67)
```

Returns the ASCII character specified by the numerical value of <expr>. In the example X$ = "C".

LEFT$	line number <string> = LEFT$(<string>,<expr>)

```
100 X$=LEFT$("COMMODORE",3)
```

Returns a substring of the <string> from the first character to the <expr>th character. In the example, X$ = "COM".

LEN	line number <var> = LEN(<string>)

```
100 X=LEN("COMMODORE")
```

Returns the number of characters contained in <string>. In the example, X = 9.

MID$	line number <string> = MID$(<string>,<expr1>, <expr2>)

```
100 X$=MID$("NOW IS THE TIME",5,6)
```

Returns a substring of <string> which begins with the character specified by <expr1> and has a length of <expr2> characters. In the example, X$ = "IS THE".

RIGHT$	line number <string> = RIGHT$(<string>,<expr>)

```
100 X$=RIGHT$("COMMODORE 64",2)
```

Returns a substring of <string> consisting of the rightmost characters specified by <expr>. In the example, X$ = "64".

STR$	line number <string> = STR$(<expr>)

```
100 X$=STR$(24.07)
```

Converts the <expr> to a string. In the example, X$="24.07".

TIME$

line number TIME$="hhmmss"
line number <string>=TIME$

```
100 TIME$="123050"
200 NOW$=TIME$
```

Sets or reads the internal timer. When TIME$ appears on the left side of an expression, the time will be set to a six digit string representing hours, minutes and seconds. The current or elapsed time can be read by using TIME$ on the right side of an expression. In the example, line 100 sets TIME$ to 12 hours, 30 minutes and 50 seconds. Line 200 sets NOW$ to the current time.

VAL

line number <var>=VAL(<string>)

```
100 X=VAL("365 DAYS")
```

Converts the <string> to a real or integer variable. The conversion will terminate once a non-numeric character is encountered. In the example, X=365.

B.7 BASIC and Disk Commands

DIRECTORY

LOAD "$",8
LIST

Prints a list on the screen of all the files on a diskette.

FORMAT

OPEN 15,8,15,"NEW:<name>,<id>"

```
OPEN 15,8,15,"NEW:MY PROGRAMS,HN"
```

Formats a blank diskette so that it can be used. <name> specifies the name of the diskette and <id> is a 2 character identification code. In the example, the diskette will be formatted with the name MY PROGRAMS and the identification code HN.

LIST

LIST {<line number>} {-<line number>}

```
LIST
LIST 300
LIST 1000-2000
```

Lists lines of the program in memory to the screen. Optionally a line number or range of line numbers may be specified. In the first example, the entire program will be listed. In the second example, line 300 only will be listed. In the third example, the lines 1000 to 2000 inclusive will be listed.

LOAD

LOAD "<file>",8

```
LOAD "SNOW WHITE",8
```

Loads the specified file from the disk drive (unit 8) into memory. The current program in memory will be erased. If an asterisk (*) is specified in place of the filename, the first program in the diskette's directory will be loaded. In the example, the program SNOW WHITE will be loaded into memory from the disk drive.

NEW

NEW

Erases the program and variables currently in memory. Used to clear memory before writing a new program.

RENAME

OPEN 15,8,15,"RENAME:<file1>=<file2>"

```
OPEN 15,8,15,"RENAME:PROGRAM 1=HEART BEATS"
```

Changes the name of <file1> to <file2> on a diskette. In the example, PROGRAM 1 will be renamed HEART BEATS on the diskette in the disk drive.

RUN

RUN {line number}

```
RUN
RUN 100
```

Executes the program in memory. If a line number is specified, the program is executed beginning with the specified line. In the first example, the program in memory will be executed from the beginning. In the second example, the program in memory will be executed starting with line 100.

SAVE

SAVE "<file>",8
SAVE "@0: <file>",8

```
SAVE "PICKLES",8
SAVE "@0:TOMATOES",8
```

Saves the program currently in memory on diskette as the file specified. To replace an existing file on the diskette with the program in memory, "@0:" must be included before the filename. In the first example, the program in memory will be saved with the new name PICKLES on diskette. The second example will replace the file TOMATOES with the program in memory.

SCRATCH OPEN 15,8,15,'SCRATCH:<file>'

```
OPEN 15,8,15,"SCRATCH:BUTTERFLIES"
```

Erases a file from a diskette. In the example, the file BUTTERFLIES will be erased from the diskette in the disk drive.

VERIFY VERIFY "<file>",8

```
VERIFY "MATH DRILL",8
```

Checks the specified file on the diskette against the program in memory to VERIFY that it was SAVEd correctly. In the example, the program MATH DRILL will be located on the diskette and verified.

B.8 Special Keys

CLEAR/HOME KEY SHIFT CLR
HOME

The HOME key moves the cursor to the home position in the upper left corner of the screen. SHIFT CLR not only HOMEs the cursor but also erases the screen.

COMMODORE KEY

The COMMODORE key is used in conjunction with the other keys on the computer's keyboard to print the graphic character on the left front of the key. The COMMODORE key used in conjunction with the SHIFT key changes the screen to lowercase mode. To execute a COMMODORE sequence, hold down the COMMODORE key while simultaneously pressing the other key.

CONTROL KEY

CTRL

The CONTROL key is used in conjunction with the number keys to set the color of the text and graphics on the screen. To execute a CONTROL sequence, hold the CONTROL key down while simultaneously pressing the number key. The available colors are:

1 black	5 purple	9 reverse on
2 white	6 green	0 reverse off
3 red	7 blue	
4 cyan	8 yellow	

CURSOR KEYS

CRSR ↑ (up)
CRSR ↓ (down)
CRSR ← (left)
CRSR → (right)

The two keys on the computer's keyboard marked CRSR are used to edit programs. Combined with the SHIFT key, the two CRSR keys can move the cursor up, down, left and right. To edit a line, position the cursor at the desired spot, make the correction, and depress the RETURN key.

INSERT/DELETE KEY

SHIFT INST
DEL

The INSERT/DELETE key is used in editing. SHIFT INST moves all text to the right of the cursor over to allow the insertion of one character. DEL deletes the character to the left of the cursor.

RESTORE KEY

RUN/STOP RESTORE

The RESTORE key when pressed simultaneously with the RUN/STOP key will clear the screen, home the cursor, and restore the screen to the standard background, border and text colors.

RUN/STOP KEY

SHIFT RUN
STOP

SHIFT RUN loads the first program found on the cassette tape and runs it. STOP terminates the execution of a program and indicates the line number where the break took place.

SHIFT KEY

SHIFT

In upper case mode, the SHIFT key is used in conjunction with the other keys on the computer's keyboard to print the graphic character on the right front of the key. In lower case mode, the SHIFT key is used in conjunction with the other keys to print the upper case of the character typed. To execute, hold either SHIFT key down while simultaneously pressing the other key.

B.9 Graphics Statements

Background Color

line number POKE 53281,<expr>

Changes the color of the background. Values of <expr> and their respective colors are:

0	black	8	orange
1	white	9	brown
2	red	10	light red
3	cyan	11	gray 1
4	purple	12	gray 2
5	green	13	light green
6	blue	14	light blue
7	yellow	15	gray 3

Border Color

line number POKE 53280,<expr>

Changes the color of the border. <expr> represents one of the color codes (0 to 15) from the table above (see "Background Color").

Text Color

line number POKE <expr1>,<expr2>

Sets the color of the specified screen character location. <expr1> is the screen memory location (from 55296 to 56295). Assuming a horizontal *x*-axis and a vertical *y*-axis with 0,0 in the upper left corner of the screen, <expr1> can be represented for any X and Y value by:

line number POKE 55296 + Y * 40 + X, <expr2>

<expr2> must be one of the color codes from 0 to 15 from the table above (see "Background Color").

Graphics

line number POKE <expr1>,<expr2>

Plots the character or graphic specified by the code <expr2> at the screen memory location specified by <expr1> (1024 to 2023). Assuming a horizontal *x*-axis and a vertical *y*-axis with 0,0 in the upper left corner of the screen, <expr1> can be represented for any X and Y by:

line number POKE 1024 + Y * 40 + X,<expr2>

<expr2> is a screen character or graphics code from the table that follows.

<expr2>	Respective ASCII Character
0 to 31	64 to 95
32 to 63	32 to 63
64 to 95	96 to 127
96 to 127	160 to 191

B.10 ASCII Character Codes

The following codes are used in the CHR$ and ASC functions:

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0		7	SWITCH TO LOWER CASE	14		21
	1	DISABLES SHIFT C=	8		15		22
	2	ENABLES SHIFT C=	9		16		23
	3		10	CRSR	17		24
	4		11	RVS ON	18		25
WHT	5		12	CLR HOME	19		26
	6	RETURN	13	INST DEL	20		27

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
RED	28	<	60	£	92	▒	124
CRSR →	29	=	61	]	93	│	125
GRN	30	>	62	↑	94	π	126
BLU	31	?	63	←	95	◥	127
SPACE	32	@	64	─	96		128
!	33	A	65	♠	97	♠	129
“	34	B	66	│	98		130
#	35	C	67	─	99		131
$	36	D	68	─	100		132
%	37	E	69	▔	101	f1	133
&	38	F	70	─	102	f3	134
·	39	G	71	▏	103	f5	135
(	40	H	72	▕	104	f7	136
)	41	I	73	╮	105	f2	137
*	42	J	74	╰	106	f4	138
+	43	K	75	╯	107	f6	139
,	44	L	76	└	108	f8	140
−	45	M	77	╲	109	SHIFT RETURN	141
.	46	N	78	╱	110	SWITCH TO UPPER CASE	142
/	47	O	79	┌	111		143
0	48	P	80	┐	112	BLK	144
1	49	Q	81	●	113	CRSR ↑	145
2	50	R	82	▁	114	RVS OFF	146
3	51	S	83	♥	115	CLR HOME	147
4	52	T	84	▏	116	INST DEL	148
5	53	U	85	╭	117	╭	149
6	54	V	86	╳	118	╳	150
7	55	W	87	○	119	○	151
8	56	X	88	♣	120	♣	152
9	57	Y	89	▕	121	▕	153
:	58	Z	90	♦	122	♦	154
;	59	[	91	┼	123	┼	155

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
PUR	156	▏	165	┐	174	▔	183
← CRSR	157	▒	166	▂	175	▀	184
YEL	158	▕	167	┌	176	▃	185
CYN	159	▄	168	┴	177	▕	186
SPACE	160	◤	169	┬	178	▖	187
▌	161	▐	170	┤	179	▝	188
▄	162	├	171	▎	180	┘	189
▔	163	▗	172	▍	181	▘	190
▁	164	└	173	▐	182	▚	191

CODES	**192-223**	**SAME AS**	**96-127**
CODES	**224-254**	**SAME AS**	**160-190**
CODE	**255**	**SAME AS**	**126**

Appendix C

Answers to Selected Questions and Problems

Chapter 1

Think About This (for Fun)

One word.

Text Questions

Section 1.5.3. The output would be close packed (printed with no separating spaces).

Section 1.5.5. The blank space is needed to separate the comma from the name (value of N$). Otherwise, the comma and name would be close packed, as in HOWDY,SAMMY.

Posers and Problems

1.
```
10 PRINT "Hello"
20 PRINT "What's your height in inches";
30 INPUT H
40 M = 2.54 * H
```

```
50 PRINT "You are " M " centimeters tall!"
60 END
```

2. 25, 4, 6, .666667, 3
3. So that the text and variable values will not be close packed.
   The variable names are *not* enclosed in quotation marks; the text of the PRINT statement is.
4. The semicolon close packed the "?" printed by execution of the INPUT statement (60).
5. See Section 2.4 of Chapter 2.
6.
```
NAME      SCORE      AVERAGE
←10 SPACES→←10 SPACES→
```
7. Enter and RUN the program.

*8.
```
10 PRINT "DEGREES CELSIUS";
20 INPUT C
30 F = (C * 9/5) + 32
40 PRINT C " DEGREES C = " F " DEGREES F."
50 END
```

*9.
```
10 PRINT "HOW MANY CUPS";
20 INPUT C
30 PRINT "HOW MANY OUNCES";
40 INPUT Z
50 T = (8 * C) + Z
60 PRINT C " CUPS " Z " OUNCES = " T " TOTAL OUNCES."
70 END
```

*10.
```
10 PRINT "FIRST NAME";
20 INPUT F$
30 PRINT "LAST NAME";
40 INPUT L$
50 PRINT "HELLO, " F$ " " L$ "!"
60 END
```

Chapter 2

Think About This (for Fun)

A chair, a bed, and a toothbrush.

Text Questions

Section 2.3.
```
INT(10 * .999999999 + 1) = 10
INT(10 * .01 + 1) = 1
Range of INT((100 * RND(0) + 1) * 10)/10 =
 100.0 to 1.0
```

```
Range of INT(9901 * RND(0) + 100)/100 = 100.00 to 1.00
Range of INT(91 * RND(0) + 5) = 95 to 5
```

Posers and Problems

1. R is for NUMERIC INPUT (years)
   R$ is for STRING INPUT (state name)
2. There is no real difference. Random output would still be given. If X is 1, then 440 is PRINTED, and so on.
3. Change the text of PRINT statements 400, 420, and 440.
4. Change PRINT statements 200 and 210 to ask for your age; change the "39" in statements 230–240 to your age.

5.
```
380 X = INT(5 * RND(0) + 1)
390 ON X GOTO 400,420,440,452,458
452 PRINT "Ho-ho!  Not THAT old!"
454 GOTO 140
456 PRINT "Come down some!"
458 GOTO 140
```

6.
```
70 PRINT "What state is the third"
80 PRINT "largest by land area";
100 IF R$ = "CALIFORNIA" OR R$ = "California" THEN 130
110 PRINT "No, it's California!"
130 PRINT "Right! Where the oranges grow!"
```

7.
```
  5 INPUT "What's your first name";F$
510 PRINT,"Bye-bye, " F$ "!"
```

9.
```
X = INT (151 * RND(0) + 50)
```

10. 29 to 5, inclusive

***11.**
```
10 PRINT "WHAT'S YOUR HEIGHT IN INCHES";
20 INPUT H
30 IF H > 72 THEN PRINT "TALL" : GOTO 60
40 IF H < 60 THEN PRINT "SHORT" : GOTO 60
50 PRINT "AVERAGE"
60 END
```

***12.**
```
10 PRINT "ENTER THREE SIDES (SEPARATED BY COMMAS),"
20 PRINT "WITH THE LONGEST SIDE ENTERED LAST"
30 INPUT A,B,C
40 IF A > C OR B > C THEN
       PRINT "LONGEST SIDE NOT ENTERED LAST!" : GOTO 10
50 IF (C ↑ 2) = ((A ↑ 2) + (B ↑ 2)) THEN
       PRINT "THAT IS A RIGHT TRIANGLE" : GOTO 70
60 PRINT "THAT IS NOT A RIGHT TRIANGLE"
70 END
```

```
*13. 10 INPUT "ENTER ANY NUMBER, 1-10, INCLUSIVE";N
     20 A$ = "WAS ENTERED."
     30 IF N = 3 THEN PRINT "THREE " A$ : GOTO 70
     40 IF N = 6 THEN PRINT "SIX " A$ : GOTO 70
     50 IF N = 9 THEN PRINT "NINE " A$ : GOTO 70
     60 PRINT "NEITHER 3, 6, OR 9 " A$
     70 END
```

Chapter 3

Think About This (for Fun)

In

Text Questions

Section 3.2.1. Yes, if the total character count does not exceed 80.

Section 3.2.2. The semicolons in statement 110 close pack the display.
Each READ assigned a new value to variable N. The value of variable X is increased by the value of N each time statement 70 is executed.

Section 3.2.3. After statement 30 is executed, the data pointer is "past" the last data element.
The error was caused by no *data* present to be *read* (the data pointer is "past" the last element).
Statement 50 increases X by 1 each time it is executed. When X is equal to 2 (statement 70), transfer is to statement 100.
The program would endlessly READ, PRINT, and RESTORE.

Section 3.3.2. Enter the same score for *both* the maximum *and* minimum scores.

Posers and Problems

1.
```
10 FOR Y = 1 TO 10

20 READ Q$,A,C$
30 PRINT Q$;C$;
50 IF A = R THEN 70

DELETE 60-70
```

(or)

```
55 RESTORE
```

(or)

```
10 DATA 4,5,6,7
```

2.
```
40 PRINT S$,S
50 NEXT I
60 END
```

3.
```
 5 T = 0
42 T = T + S
44 X = X + 1
55 PRINT "THE AVERAGE SCORE IS " T/X "!"
```

4. LOAD, RUN, and LIST program A354 from the text diskette.

5. FOR X = 10 TO 1 STEP −1 starts at 10 and "counts" the loop to 1 in increments of −1.
   The commas in statements 40 and 90 tab ten spaces before printing.
   The ";" in statement 60 close packs the "tails" (*).
   The 80 PRINT statement cancels the close packing effect of the semicolon in statement 60.

6.
```
10 PRINT "CELSIUS","FAHRENHEIT"
20 PRINT "-------","----------"
30 FOR C = 0 TO 100 STEP 5
40    F = 32 + (C * 9/5)
50    PRINT C,F
60 NEXT C
70 END
```

7.
```
10 FOR I = 1 TO 10
20    PRINT I " CUBED IS " I ↑ 3
30 NEXT I
40 END
```

8. The *minimum* changes would be:

```
LOAD "PROGRAM 5A",8
240 PRINT "MULTIPLICATION DRILL"
510 A = N1 * N2
530 PRINT N1 " x " N2 " = ";
SAVE "PROGRAM 5M",8
```

9. The *minimum* changes would be:

```
LOAD "PROGRAM 5A",8
240 PRINT "DIVISION DRILL"
430 N1 = INT(141 * RND(0) + 4)
440 N2 = INT(141 * RND(0) + 4)
445 IF N1/N2 <> INT(N1/N2) THEN 430
510 A = N1 / N2
530 PRINT N1 " / " N2 " = ";
SAVE "PROGRAM 5D",8
```

Chapter 4

Think About This (for Fun)

A 50-cent piece and a nickel (one of the coins is not a nickel—although the other one is!).

Text Questions

Section 4.2.1. N$(3) = PHIL; S(4) = 35; the two lists would be printed.
The lists would be printed, but in reverse order (4 to 1).

Section 4.2.2. S(1,1) = 95; S (3,2) = 93
For the complete program in Section 4.2.2, LOAD, RUN, and LIST program A422 on the text diskette.
The commas in statements 65 and 81 are needed to tab before printing the first and second scores. The PRINT in 95 is needed to cancel the tab effect of statement 81.

Posers and Problems

1.
```
10 REM N$()=NAME;S()=SEM. AVE.;F()=FINAL EXAM
20 DIM N$(20),S(20),F(20)
30 FOR I = 1 TO 20
40    READ N$(I),S(I),F(I)
50    PRINT N$(I),S(I),F(I)
60 NEXT I
 .
 .
 .
1000 REM DATA FOR 20 STUDENTS AND THEIR SCORES
1010 DATA "JONES",80,82,etc.
 .
 .
 .
2000 END
```

2.
```
10 REM N$()=NAME,S( , )=STUDENT SCORES
20 DIM N$(25),S(25,3)
30 FOR I = 1 TO 25
40    READ N$(I)
50       FOR J = 1 TO 3
60          READ S(I,J)
70       NEXT J
80 NEXT I
 .
 .
 .
```

```
1000 REM DATA FOR 25 STUDENTS, EACH WITH 3 SCORES
1010 DATA "ABEL,95,80,88, etc.
```

4. Three states would be randomly selected (with a random chance that one would be repeated).
5. Three states would be randomly selected without any state being repeated.
6. Five states would be printed without any repetition. *Then* the program becomes an endless loop, trying to find the *sixth* state not yet printed (FOR K = 1 TO 6) when only five states were given.
7.

```
   5 F = 0
  25 H$ = "STEPHEN F.   --?--"
  55 H$ = "DICK AND JANE'S DOG."
1010 IF R$ = A$ THEN PRINT "GREAT" : RETURN
1012 IF F  = 0 THEN PRINT "HINT:   " H$:F = 1:GOTO
    1000
1014 PRINT "NOPE...IT'S " A$ : F = 0
```

8. LOAD, RUN and LIST program A458 on the next diskette.
9.

```
10 DIM Z(10)
20 FOR I = 1 TO 4
30    X = INT(10 * RND(0) + 1)
40    IF Z(X) = 1 THEN 30
50    Z(X) = 1
60    PRINT X
70 NEXT I
80 END
```

10. LOAD and LIST DRILL MENU from the disk.

    Add the necessary "POKE" statements that will RUN "DRILL MENU" as the last executable statements in PROGRAM 5, PROGRAM 5A, PROGRAM 5M, and PROGRAM 5D. (Substitute the names you used if different than these.)

    The DRILL MENU program must be SAVEd prior to RUNning because if it successfully executes, it will LOAD and RUN another program. This erases the DRILL MENU program from memory.

Chapter 5

Think About This (for Fun)

The man opened one carton, took one package, opened it, and then dropped one cigarette overboard. This made the raft a cigarette lighter!

Posers and Problems

1.
```
 .
 .
 .
200 INPUT R$
210 IF R$ = A$ THEN PRINT "VERY GOOD"
    :  IF F = 0 THEN C = C + 1 : RETURN
220 IF F = 0 THEN PRINT "HINT:  " H$ : F = 1 : GOTO
    200
224 PRINT "THE ANSWER IS " A$ : F = 0 : RETURN
 .
 .
 .
```

2. See statement 220 in the above program fragment.

3.
```
  10 REM Q$()= QUESTION; A$()=ANSWER; Z()=FLAG
  20 DIM Q$(10),A$(10),Z(10)
  30 FOR I = 1 TO 10
  40    READ Q$(I),A$(I)
  50 NEXT I
  60 FOR Q = 1 TO 5
  70    X = INT(10 * RND(0) + 1)
  80    IF Z(X) = 1 THEN 70
  90    Z(X) = 1
 100    PRINT Q$(X)
        .
        .
        .
1000 REM DATA FOR 10 QUESTIONS, ANSWERS
1010 DATA "QUESTION 1","ANSWER 1", etc.
        .
        .
        .
2000 END
```

4. LOAD, LIST, and RUN program A544 from the text diskette. (*Note:* Very few REM statements are included in this program. Try to mentally interpret its execution from the LISTing.)

Chapter 6

Think About This (for Fun)

There are six F's (the word *of* is often overlooked).

Posers and Problems

1.
```
LOAD "PROGRAM 10",8
125 REM  C()  – The count for a given score
265 DIM C(100)
395 C(S(I)) = C(S(I)) + 1
712 FOR I = 100 TO 1 STEP -1
713    IF C(I) = 0 THEN 716
715    PRINT "SCORE = " I, "COUNT = " C(I)
716 NEXT I
717 IF Z$ <> "Y" THEN 720
718 PRINT 4 : CLOSE4
SAVE "PROGRAM 10",8 (OPTIONAL)
```

2.
```
180 DIM A$(15),Q$(15),Z(15)
310 FOR I = 1 TO 15: . . .
350 FOR I = 1 TO 10
620-???  (DATA for 15 question/answer pairs of your choice)
```

Chapter 7

Think About This (for Fun)

Bookkeeper

Text Questions

Section 7.4.2. For five choices, make the following modifications:

```
270 DIM A$(5),R$(5)
5050 FOR I = 1 TO 5 : . . .
5110 . . . (1 - 5) . . .
5150 . . . OR R > 5  . . .
```

Then add additional data elements for each fifth choice and its response.

Chapter 8

Think About This (for Fun)

6

Posers and Problems

1.
```
30 FOR I = 1 TO 12
70 IF PW$ = "COMMODORE 64" THEN . . .
```

Chapter 9

Think About This (for Fun)

Sleeplessness.

Chapter 10

Think About This (for Fun)

50 1/2 + 49 38/76 = 100

Chapter 11

Think About This (for Fun)

The frog would make 28 jumps (after 27 jumps, the frog is 3 feet from the top of the well; one more jump of 3 feet is needed).

Appendix D

Listings of Solution Programs to Posers and Problems

Chapter 3 Posers and Problems Question 4

```
10 REM  ===============
20 REM   PROGRAM 'A354'
30 REM  ===============
40 REM
50 REM  MODIFIED DIRECTLY FROM PROGRAM 4
60 REM
80 REM  ===============
90 REM  VARIABLE DICTIONARY
100 REM  ==============
110 REM  N$- HYPOTHETICAL NAME
120 REM  H$- HYPOTHETICAL HAIR COLOR
130 REM  E$- HYPOTHETICAL EYE COLOR
140 REM  C$- EYE COLOR SOUGHT
150 REM  I - LOOP COUNTER
160 REM  F - COUNTER FOR THE NUMBER OF
    MATCHES FOUND
170 REM  ===========
180 REM  SET THE COUNTER  TO ZERO,
    CLEAR
190 REM  THE SCREEN, AND GET THE RANGE
    SOUGHT.
200 REM  ===========
210 F = 0
220 PRINT "♡"
230 PRINT TAB(5) "THE EYE COLOR SOUGHT
    IS";
240 INPUT C$
270 PRINT "NAMES WITH EYE COLOR " C$ "
    ARE:"
280 PRINT "NAME","EYE COLOR","HAIR
    COLOR"
290 PRINT "----","--- -----","----
    -----"
300 REM  ===============
310 REM  DO THE SEARCH LOOP
320 REM  ===============
330 FOR I = 1 TO 50
340 READ N$,E$,H$
```

```
350 REM   ===END OF DATA LIST?===
360 IF N$ = "X" THEN 460
370 REM  ===DO THE COLORS MATCH?===
380 IF C$ <> E$ THEN 420
390 PRINT N$,E$,H$
400 REM  ===COUNT THE MATCHES FOUND===
410 F = F + 1
420 NEXT I
430 REM  ===============
440 REM  END OF CURRENT SEARCH
450 REM  ===============
460 PRINT "THIS SEARCH FOUND " F "
    MATCH(ES)."
470 REM  ===GIVE OPTION TO DO ANOTHER
    SEARCH===
480 PRINT "DO YOU WISH ANOTHER SEARCH
    (Y OR N)";
490 INPUT Z$
500 IF Z$ <>"Y" THEN 1000
510 RESTORE
520 GOTO 210
530 REM  ==============
540 REM     DATA LIST
550 REM  ==============
560 DATA "KAY","GREEN","BROWN","TRACY",
    "BLUE","BROWN","HERB","BLUE",
    "BALD"
570 DATA "JANE","BROWN","BLACK","SUE",
    "BROWN","BLOND","BUCK","BLUE",
    "BLOND"
580 DATA "KARYN","GREEN","BROWN","SAM",
    "BLUE","BLOND"
590 DATA "ANNE","HAZEL","AUBURN"
600 REM ===ROOM FOR MORE DATA===
999 DATA  "X","X","X"
1000 PRINT "*** SEARCH COMPLETED ***"
1010 END
```

Chapter 4 Text Question Section 4.2.2

```
5 REM  ===PROGRAM A422===
10 DATA  "CHUCK",95,80
20 DATA  "MARY",80,82
30 DATA  "PHIL",95,93
40 DATA  "JEANNIE",35,98
45 PRINT "NAME","TEST1","TEST2"
46 PRINT "----","-----","-----"
50 FOR I = 1 to 4
60 READ N$(I)
65 PRINT N$(I),
70 FOR J = 1 TO 2
80 READ S(I,J)
81 PRINT S(I,J),
82 REM  T(I) = CUMULATIVE TOTAL EACH
   STUDENT'S SCORE
84 T(I) = T(I) + S(I,J)
86 REM T = CUMULATIVE TOTAL ALL SCORES
88 T = T + S(I,J)
90 NEXT J
94 REM SKIP A LINE PRIOR TO PRINT OF
   NEXT NAME
98 PRINT
100 NEXT I
110 PRINT
120 FOR I = 1 TO 4
130 PRINT N$(I) "'S AVERAGE IS"
          T(I) / 2
140 NEXT I
150 PRINT "THE CLASS AVERAGE IS" T / 8
160 END
```

Chapter 4 Posers and Problems Question 8

```
10 REM   PROGRAM NAME:  A458
20 REM  ====================
30 REM  CREATES FIVE RANDOM SENTENCES
40 REM  FROM 3 LISTS OF 7 SUBJECTS,
50 REM  VERBS, AND DIRECT OBJECTS.
60 REM  ====================
70 REM  VARIABLE DICTIONARY
80 REM  ====================
90 REM   D$( ) - 7 DIRECT OBJECTS
100 REM  S$( ) - 7 SUBJECTS
110 REM  V$( ) - 7 VERBS
120 REM  X,Y,Z - RANDOM NUMBERS, 1-7,
130 REM  INCLUSIVE, FOR A SUBJECT
140 REM  VERB, AND DIRECT OBJECT
150 REM  ===================
160 DIM S$(7),V$(7),D$(7)
162 PRINT "♡" : FOR I = 1 TO 11 : PRINT:
    NEXT I : PRINT,"A BIT OF NONSENSE"
164 FOR I = 1 to 1500 : NEXT I : PRINT
    "♡"
170 REM  ===READ THE SUBJECTS===
180 FOR I =1 TO 7
190 READ S$(I)
200 NEXT I
210 REM  ===READ THE VERBS===
220 FOR I = 1 TO 7
230 READ V$(I)
240 NEXT I
250 REM  ===READ THE DIRECT OBJECTS===
260 FOR I = 1 TO 7
270 READ D$(I)
280 NEXT I
290 REM  ===PRINT 5 RANDOM SENTENCES===
300 FOR I = 1 TO 5
305 PRINT
310 X =  INT (7 * RND (0) + 1)
320 Y =  INT (7 * RND (0) + 1)
330 Z =  INT (7 * RND (0) + 1)
340 PRINT S$(X) " " V$(Y) " " D$(Z) "!"
350 NEXT I
352 PRINT : PRINT : PRINT : INPUT "DO
    IT AGAIN (Y OR N)";Z$
354 IF Z$ = "Y" THEN PRINT "♡" : GOTO
    300
360 REM  ===DATA FOR SUBJECTS===
370 DATA  "JIM","THE DOG","THE CAT"
380 DATA  "SUE","THE GOLD FISH","BILL"
390 DATA  "JACK AND JILL"
400 REM  ===DATA FOR VERBS===
410 DATA  "HATED","LOVED","WORE",
    "KISSED"
420 DATA  "TOSSED","CAUGHT","ATE"
430 REM  ===DATA FOR DIRECT OBJECTS===
440 DATA  "THE BALL","THE HILL","THE
    HAT"
450 DATA  "THE COOKIES","THE BAT","THE
    HAT"
460 DATA  "THE MICROCOMPUTER"
470 END
```

Chapter 4 Posers and Problems Question 10

```
10 REM  DRILL MENU (CHAPTER 4)
20 REM  ================
30 REM  PROGRAM DEMONSTRATES HOW A
   'MENU' OF PROGRAMS
40 REM  MAY BE PROVIDED TO A USER.
   USING 'POKE'
50 REM  STATEMENTS, ONE PROGRAM WILL
   AUTOMATICALLY
60 REM  'LOAD' AND 'RUN' THE PROGRAM
   DEFINED AS
70 REM  THE STRING P$(C).
72 REM
74 REM  (THIS PROGRAM WAS MADE BY
   MODIFYING
76 REM  'MENU' PROGRAM
78 REM
80 REM  ===============
90 REM
100 PRINT "♡" : PRINT : PRINT
110 PRINT TAB(6) "M A T H  D R I L L
    M E N U" : PRINT
120 PRINT,"YOUR OPTIONS ARE:" : PRINT
130 PRINT,"1. MATH SUBTRACTION"
140 PRINT,"2. MATH ADDITION"
142 PRINT,"3. MATH MULTIPLICATION"
144 PRINT,"4. MATH DIVISION"
150 PRINT,"5. STOP"
160 PRINT : PRINT, : INPUT "YOUR CHOICE
    IS (1-5):";C
164 IF C < 1 OR C > 5 THEN 160
170 IF C = 5 THEN PRINT "♡" : PRINT
    "SELECTIONS COMPLETED" : END
180 REM
190 REM ===STORE VALUES IN P$()===
200 REM
210 FOR I = 1 TO 4 : READ P$(I) : NEXT I
220 PRINT "♡"
230 REM
240 REM ===LOAD AND RUN USING
    'POKES'===
250 REM
260 PRINT "LOAD ";CHR$(34);P$(C);
    CHR$(34);",8"
270 POKE 631,19 : POKE 632,17 : POKE
    633,13
280 POKE 634,82 : POKE 635,85 : POKE
    636,78 : POKE 637,13 : POKE 198,7
290 REM
300 REM ===DATA FOR PROGRAM NAMES TO
    LOAD AND RUN===
310 REM
320 DATA "PROGRAM 5","PROGRAM
    5A","PROGRAM 5M","PROGRAM 5D"
```

Chapter 5 Posers and Problems Question 4

```
10 REM   ===A544===
20 REM
30 REM  ONE SOLUTION TO PROBLEM 5.4.4
40 REM
50 REM  DIMENSION VARIABLES
60 REM
70 DIM Q$(15),A$(15),H$(15),Z(15),
   C$(3),W$(3)
```

```
72 REM
60 REM  STORE QUESTIONS, ANSWERS, HINTS
90 REM
100 REM I = 1 TO 15 : READ Q$(1),
    A$(I),H$(I) : NEXT I
110 REM
120 REM STORE CORRECT, INCORRECT
    FEEDBACK
130 REM
140 FOR I = 1 TO 3 : READ C$(I),W$(I) :
    NEXT I
150 REM
160 REM CLEAR SCREEN, SHOW TITLE
170 REM
180 PRINT "♡" : FOR I = 1 TO 11 : PRINT
    : NEXT I
190 PRINT,"EIGHT RANDOM QUESTIONS" :
    FOR I = 1 TO 1500 : NEXT I
200 REM
210 REM EIGHT QUESTION LOOP
220 REM
230 FOR Q = 1 TO 8
240 PRINT "♡" : FOR I = 1 TO 6 : PRINT
    : NEXT I : F = 0
250 REM
260 REM RANDOMLY SELECT QUESTION NUMBER
    FROM LIST
270 REM
260 J = INT(15 * RND(0) + 1)
290 REM
300 REM CHECK FOR FIRST SELECTION OF
    THIS NUMBER
310 REM
320 IF Z(J) = 1 THEN 280
330 REM
340 REM 'FLAG' THE NUMBER SELECTED
350 REM
360 Z(J) = 1
370 REM
360 REM PRINT THE QUESTION, GET AN
    ANSWER
390 REM
400 PRINT : PRINT Q$(J); : INPUT R$ :
    PRINT
410 REM
420 REM GET A RANDOM NUMBER FOR
    FEEDBACK
422 REM
430 R = INT(3) * RND(0) + 1)
432 REM
434 REM CHECK ANSWER AND RESPOND
    ACCORDINGLY
436 REM
440 IF R$ = A$(J) THEN PRINT,C$(R) :
    C = C + 1 : GOTO 500
450 IF F = 0 THEN PRINT,W$(R) " HINT:"
    : PRINT,H$(J) : F = 1 : GOTO 400
460 REM
470 REM GIVE ANSWER ON SECOND MISS
480 REM
490 PRINT "A CORRECT ANSWER IS: " A$(J)
500 PRINT : INPUT "PRESS RETURN TO
    CONTINUE";Z$
510 NEXT Q
520 PRINT "♡" : FOR I = 1 TO 11 : PRINT
    : NEXT I
530 PRINT "YOU ANSWERED" C "CORRECTLY."
540 END
550 REM
560 REM 15 QUESTIONS, ANSWERS, HINTS
570 REM
580 DATA "LAST NAME OF 'BOLERO'
    COMPOSER","RAVEL","SWEATERS
    CAN UN-"
590 DATA "PROGRAMS ARE STORED ON A
    FLOPPY -","DISK","SOME HAVE
    SLIPPED"
600 DATA "THE LARGEST STATE
    IS","ALASKA","FROZEN BUNS"
610 DATA "THE LARGEST MAKER OF
    COMPUTERS IS","IBM","FOUNDED BY
    HOLLERITH"
620 DATA "THE FIRST 'CALCULATOR' WAS
    THE","ABACUS","ABRA-KA-DABRA"
630 DATA "THE FIRST ELECTRONIC COMPUTER
    WAS","ENIAC","JUST 'ENY' OL' NAME"
640 DATA "THE 'DISPLAY' STATEMENT IN
    BASIC","PRINT","GUTENBURG"
650 DATA "EVERY GOSUB NEEDS A",
    "RETURN","IT'S ON THE KEYBOARD"
660 DATA "DATA IN BASIC MUST BE",
    "READ","---- SAILS IN THE SUNSET"
670 DATA "A VARIABLE IS 'NUMERIC'
    OR","STRING","HANGING BY A THREAD"
660 DATA "THE 'SHOW ME' STATE IS",
    "MISSOURI","GATEWAY TO THE WEST"
690 DATA "'HARD COPY' IS OBTAINED FROM
    A","PRINTER","THE PRINCE AND THE
    PAUPER"
700 DATA "THERE'S HARDWARE, FIRMWARE
    AND","SOFTWARE","CUSHIONY-WARE"
710 DATA "A 'SEA-FARING' COMPUTER
    MAN","COMMODORE","YOU KNOW HIM
    WELL"
720 DATA "A STATEMENT SEEN BUT NOT
    EXECUTED","REM","SORRY, NO REMARKS
    HERE"
730 REM
```

```
740 REM 3 FEEDBACK SETS
750 REM
760 DATA "G R E A T","I'LL SAY THIS"
770 DATA "F A N T A S T I C","TRY THIS"
780 DATA "H O T - D O G","LOOK AT THIS"
```

Chapter 7 Test Question Section 7.3.2

```
10 REM   PROGRAM NAME: A 732
20 REM  ====================
30 REM  SHOWS SOLUTION TO QUESTION
40 REM  ON ALPHABETIZING (SORTING)
50 REM  QUESTION IN SEC. 7.3.2
60 REM  ====================
70 REM  VARIABLE DICTIONARY
80 REM  ====================
100 REM  L$ - LAST NAME (SPACE) FIRST
    NAME
110 REM  N - NUMBER OF NAMES TO SORT
120 REM  ====================
130 DIM L$(100)
140 GOSUB 1000: REM  SCREEN CLEAR/
    CENTER
150 PRINT "THIS PROGRAM WILL
    ALPHABETIZE"
160 PRINT "A LIST OF NAMES INPUT IN
    THE"
170 PRINT "SEQUENCE: LAST NAME FIRST
    NAME."
180 PRINT : PRINT "ENTER 'STOP' FOR THE
    ALPHABETIZED"
190 PRINT "LIST.  (ANY KEY TO
    CONTINUE..."
200 GET Z$: IF Z$="" THEN 200
210 GOSUB 1000
220 FOR N = 1 TO 100
230 PRINT "LAST NAME FIRST NAME";
240 INPUT L$(N)
250 IF L$(N) = "STOP" THEN 300
280 GOSUB 1000
280 NEXT N
300 N = N - 1
310 REM  ===================
320 REM  SORTING ROUTINE
330 REM  ===================
340 FOR J = 1 TO N
350 D1$ = L $(J)
370 FOR K = J - 1 TO STEP  - 1
380 IF L$(K) <  = D1$ THEN 430
380 L$(K + 1) = L$(K)
410 NEXT K
420 K = 0
430 L$(K + 1) = D1$
450 NEXT J
460 REM  ==========END OF SORT=========
470 GOSUB 1000
480 PRINT "DO YOU WANT A PRINTER COPY
    (Y OR N)?"
490 GET Z$:IF Z$="" THEN 490
482 GOSUB 1000
500 IF Z$ = "Y" THEN OPEN 4,4 : CMD4
510 PRINT : PRINT "ALPHABETIZED LIST:":
    PRINT
520 FOR I = 1 TO N
530 PRINT I "   " L$(I)
540 NEXT I
550 IF Z$="Y" THEN PRINT#4 : CLOSE 4
560 PRINT
570 PRINT "DO YOU WANT ANOTHER LIST"
580 GET Z$:IF Z$="" THEN 580
590 IF Z$ = "Y" THEN 470
600 GOTO 1500
980 REM  ========SCREEN SUBROUTINE
    =======
1000 PRINT "♡" : FOR I = 1 TO 10: PRINT
     : NEXT I: RETURN
1010 REM  ==========END OF
     ROUTINE========
1500 PRINT : PRINT "*** D O N E ***"
1515 END
```

Chapter 7 Section 7.3.3

```
260 REM  PROGRAM ISLAND
270 DIM C$(5),L$(5),R$(5),T$(5),Z(5)
280 PRINT "♡":FOR I=1 TO 8:PRINT:NEXT I
300 PRINT " D I S C O V E R
    S O M E T H I N G"
310 PRINT
320 PRINT " A B O U T   Y O U R S E L F !"
330 FOR I=1 TO 1500:NEXT I:PRINT "♡"
340 FOR I=1 TO 11:PRINT:NEXT I
350 PRINT "   O U R   S T O R Y . . ."
360 FOR I=1 TO 1000:NEXT I:PRINT "♡"
410 PRINT "THERE ARE FIVE SURVIVORS
    FROM A"
420 PRINT:PRINT "SHIPWRECK.  (CALL THEM
    PEOPLE"
430 PRINT:PRINT "A, B, C, D, AND E.) A,
    B, AND C"
440 PRINT:PRINT "ARE ON ONE ISLAND; ON
    ANOTHER"
```

```
450 PRINT:PRINT "ISLAND, WITH SHARK-
    INFESTED "
460 PRINT:PRINT "WATERS BETWEEN, ARE D
    AND E."
470 PRINT:PRINT "PERSONS A AND D ARE
    BETROTHED."
480 PRINT:PRINT "NOW PERSON 'A' WANTS
    TO BE WITH"
490 PRINT:PRINT "PERSON 'D', SO 'A'
    STARTS BUILDING"
500 PRINT:PRINT "A BOAT. (ANY KEY WILL
    CONTINUE ...)"
570 GET Z$:IF Z$="" THEN 570
580 PRINT "♡":PRINT "BUT 'A' GETS TO
    THE POINT IN THE"
590 PRINT:PRINT "BOAT BUILDING WHERE
    HELP IS NEEDED."
600 PRINT:PRINT "SO, 'A' GOES TO 'B'
    AND ASKS FOR"
610 PRINT:PRINT "SOME HELP. NOW 'B' IS
    BUSILY"
620 PRINT:PRINT "WORKING ON SOME SAL-
    VAGED RADIO PARTS"
630 PRINT:PRINT "AND FEELS THAT CON-
    TACTING SOMEONE"
640 PRINT:PRINT "IS MUCH MORE IMPORTANT
    THAN A'S BOAT."
650 PRINT:PRINT "SO 'A' ASKS 'C' FOR
    HELP WITH THE"
660 PRINT:PRINT "BOAT BUILDING. 'C'
    READILY AGREES,"
670 PRINT:PRINT "BUT ONLY ON THE CONDI-
    TION THAT 'A'"
680 PRINT:PRINT "AND 'C' BECOME LOVERS.
    (ANY KEY)"
690 GET Z$:IF Z$="" THEN 690
700 PRINT "♡" :PRINT "WELL, 'A' THINKS
    IT OVER AND DECIDES"
710 PRINT:PRINT "THAT ANYTHING IS WORTH
    BEING WITH"
720 PRINT:PRINT "'D'.  EVENTUALLY, THE
    BOAT IS BUILT,"
730 PRINT:PRINT "AND 'A' GETS TO THE
    OTHER ISLAND."
740 PRINT:PRINT "HOWEVER, 'D' SAW WHAT
    WAS HAPPENING"
750 PRINT:PRINT "AS THE BOAT WAS BEING
    BUILT!"
760 PRINT:PRINT "'D' IS HEARTBROKEN BY
    WHAT 'A' HAS"
762 PRINT:PRINT "DONE, AND CAN NO
    LONGER LOVE 'A'.
770 PRINT:PRINT "THIS CRUSHES A'S FEEL-
    INGS TO THE"
780 PRINT:PRINT "POINT THAT SUICIDE IS
    ATTEMPTED!"
790 PRINT:PRINT "(ANY KEY FOR THE
    CLIMAX!)"
800 GET Z$:IF Z$="" THEN 800
810 PRINT "♡":PRINT "REMEMBER 'E'?
    WELL, 'E' SEES WHAT"
820 PRINT:PRINT "'A' IS DOING, AND
    STOPS THE ATTEMPT!"
830 PRINT:PRINT "'E' THEN SPENDS TIME
    COUNSELING WITH"
840 PRINT:PRINT "'A' UNTIL A SENSE OF
    WORTH IS BACK."
850 PRINT:PRINT "SOON THEREAFTER, A
    'LOOKOUT' ON A"
852 PRINT:PRINT "PASSING SHIP SEES THE
    SURVIVORS, AND"
854 PRINT:PRINT "ALL ARE RESCUED!"
860 PRINT:PRINT:PRINT:PRINT "   SO..."
870 PRINT:PRINT:PRINT "(ANY KEY...)"
880 GET Z$:IF Z$="" THEN 880
890 PRINT "♡":PRINT"  WE HAVE THE FIVE
    PEOPLE:"
900 PRINT
910 PRINT:PRINT"  A","THE BOAT BUILDER"
920 PRINT:PRINT"  B","THE RADIO
    BUILDER"
930 PRINT:PRINT"  C","THE HELPER/LOVER"
940 PRINT:PRINT"  D","THE HEARTBROKEN"
950 PRINT:PRINT"  E","THE COUNSELOR"
960 PRINT:PRINT "PLEASE RANK THESE
    PEOPLE BY LETTER"
970 PRINT "IN ORDER OF YOUR PERSONAL
    'ESTEEM'..."
980 PRINT
1020 FOR I=1 TO 5
1030 PRINTTAB(8) "RANK" I;
1040 INPUT R$(I)
1045 IF R$(I)<"A" OR R$(I)>"E" THEN
     PRINT,"* OUT OF RANGE *" : GOTO
     1030
1050 NEXT I
1060 PRINT "♡":FOR I= 1 TO
      11:PRINT:NEXT I
1070 PRINT"  HMMMMM...LET ME SEE..."
1080 FOR I=1 TO 2000:NEXT I
1120 DATA "A","B","C","D","E"
1140 FOR I=1 TO 5:READ C$(I):NEXT I
1180 DATA "HAVE SEXUAL INCLINATIONS"
1190 DATA "THINK THINGS OUT
     INTELLIGENTLY"
1200 DATA "TAKE ADVANTAGE OF ANY
     OPPORTUNITY"
1210 DATA "ARE A RIGHTEOUS PERSON"
```

```
1220 DATA "ARE A WARM AND LOVING
     PERSON"
1240 FOR I=1 TO 5:READ T$(I):NEXT I
1280 DATA "VERY OFTEN","REGULARLY",
     "SOMETIMES"
1290 DATA "EVERY SO OFTEN","ONLY VERY
     RARELY"
1310 FOR I=1 TO 5:READ L$(I):NEXT I
1370 FOR I=1 TO 5:PRINT "♡":PRINT
1380 PRINT" MY PROFILE RANKING OF YOU
     INDICATES:":PRINT
1390 PRINT:PRINT:PRINT:PRINT:PRINT
1400 PRINT,L$(I) ", YOU"
1480 FOR J=1 TO 5
1490 IF R$(I)<>C$(J) THEN 1550
1520 PRINT TAB(5) T$(J) "!"
1530 PRINT:PRINT:PRINT TAB(10);"(ANY
     KEY...)"
1540 GET Z$:IF Z$="THEN 1540
1550 NEXT J
1560 NEXT I
1570 PRINT "♡":FOR I=1 TO
     11:PRINT:NEXT I
1580 PRINT TAB(8);"SO NOW YOU KNOW..."
1590 PRINT
1600 PRINT" (BUT I'M NO
     PSYCHOLOGIST...)"
1610 END
```

Appendix E

Making Music with the Commodore 64

Commodore BASIC supports the creation of sound effects and music using the external speaker in the television or monitor. This appendix describes the various statements and procedures necessary to play music on the Commodore 64. It is included as an appendix rather than a chapter section because some of the statements require a knowledge of music that not all readers may have, and the programming steps to create a score may be tedious (i.e., if you don't have a lot of patience and you can't read music, you'd better skip this appendix).

E.1 Generating Music

The Commodore 64 has a remarkable capacity for generating sound. As usual in the computer world, this expanded capacity is associated with complexity. The Commodore has three voices that can be played individually or together in any combination to form 96 notes (8 octaves). For simplicity, this appendix limits its discussion to voice number 1 but tables including all necessary information to use all three voices are included.

Each voice requires six settings in order to generate music: volume, attack/decay rate, sustain/release rate, high frequency of note, low frequency of note, and the waveform pattern. The first three settings are executed once in a program, usually at the beginning. They set the loudness and the type of instrument being played. The last three settings define the frequency of the note and the pattern of its sound wave. These settings are repeated for each note in the song.

VOLUME SETTING

line number POKE 54296,<expr>

Sets the volume for all voices based on the value of <expr>. A value of 0 turns off the volume and a value of 15 is the loudest setting.

ATTACK/DECAY

line number POKE 54277,<expr>

Sets the attack/decay rate based on the value of <expr>. Use the following chart as a guide in choosing the attack/decay rate, the sustain/release rate, and the waveform pattern:

Instrument	Attack/ Decay	Sustain Release	Waveform
Trumpet	96	0	33
Flute	96	0	17
Harpsichord	9	0	33
Xylophone	9	0	17
Organ	0	240	33
Calliope	0	240	17
Accordion	102	0	17

SUSTAIN/RELEASE

line number POKE 54278,<expr>

Sets the sustain/release rate based on the value of <expr>. See chart above for suggested settings.

NOTE FREQUENCIES

line number POKE 54273,<expr1>
line number POKE 54272,<expr2>

Sets the frequency of the note to be played based on the following formulae:

```
expr1 = INT((frequency/2 ↑ (7-octave))
        / 256)
expr2 = INT((frequency/2 ↑ (7-octave))
        - expr 1 * 256)
```

where octave is a number from 0 to 7 representing 8 octaves and frequency is the note frequency based on the chart below.

Note	Frequency	Note	Frequency
C	34334	F#	48556
C#	36376	G	51443
D	38539	G#	54502
D#	40830	A	57743
E	43258	A#	61176
F	45830	B	64814

See Section F.4 for a BASIC program to print the values of <expr1> and <expr2> for every note in all 8 octaves.

WAVEFORM PATTERN

line number POKE 54276,<expr>

Sets the pattern for the sound wave based on the value of <expr> and begins to play the note. Two basic waveforms are used to play music: the triangle (<expr> = 17) and the sawtooth (<expr> = 33). To stop the note from playing use a value for <expr> equal to <expr> – 1 (i.e. 16 for a triangle wave and 32 for a sawtooth wave).

E.2 A Musical Example

In order to write a program to play music, you will need to include the following steps in your program:

1. Set the VOLUME (preferably at the highest setting).
2. Set the appropriate ATTACK/DECAY rate.
3. Set the appropriate SUSTAIN/RELEASE rate.
4. Compute the high and low FREQUENCY values for the note (or find them in the chart produced by the program in Section F.4). Use the two POKEs to set the note frequency.
5. Set the WAVEFORM pattern, which begins playing the note.
6. Enter a FOR-NEXT loop that sets the DURATION of the note. For example the statement 50 FOR T = 1 TO 250:NEXT T is approximately a quarter note.

7. Stop the note by poking the WAVEFORM pattern with <expr> – 1.

The following program is a short example of the sound statements described above (see if you can name that tune in one note):

```
10 POKE 54296,15              :REM SET VOLUME
20 POKE 54277,0               :REM SET ATTACK/DECAY
30 POKE 54278,240             :REM SET SUSTAIN/RELEASE
40 POKE 54273,16              :REM SET HIGH FREQUENCY
50 POKE 54272,195             :REM SET LOW FREQUENCY
60 POKE 54276,17              :REM SET WAVEFORM
70 FOR T=1 TO 250:NEXT T      :REM HOLD 1/4 NOTE
80 POKE 54276,16              :REM STOP NOTE
100 REM
110 REM CONTINUE PLAYING NOTES
120 REM
```

E.3 Let the Voices Join In

As indicated in the beginning of this appendix, three voices are available that can be used for chords and harmony. The discussion above only applies to the first voice. However, steps to making music with the other two voices are the same but the memory locations that need to be POKEd are different. The following chart will give you a summary of the memory locations for all three voices.

Function	Voice 1	Voice 2	Voice 3
Volume	54296	54296	54296
Attack/Decay	54277	54284	54291
Sustain/Release	54278	54285	54292
High frequency	54273	54280	54287
Low frequency	54272	54279	54286
Waveform	54276	54283	54290

E.4 Generating a Note Frequency Table

The BASIC program below will print a table of high and low frequency values and play each of the 12 notes in the 8 octaves. This printed table may aid in the determination of values to be POKEd for short songs. Programs that play longer songs will be more efficient if they use the formulae in Section F.1. (This program appears on the diskette available for this book as NOTE TABLE.)

```
10 POKE 54296,15
20 POKE 54277,0
30 POKE 54278,240
40 FOR OCT = 0 TO 7
50 FOR N = 1 TO 12
60 READ NTE,N$
70 E1=INT((NTE/2↑(7-OCT))/256)
80 E2=INT((NTE/2↑(7-OCT))-E1*256)
90 PRINT OCT,N$,E1,E2
100 POKE 54273,E1
110 POKE 54272,E2
120 POKE 54276,17
130 FOR T=1 TO 250: NEXT T
140 POKE 54276,16
150 NEXT N
160 RESTORE
170 NEXT OCT
180 END
190 DATA 34334,"C",36376,"C#",38539,"D"
200 DATA 40830,"D#",43258,"E",45830,"F"
210 DATA 48556,"F#",51443,"G",54502,"G#"
220 DATA 57743,"A",61176,"A#",64814,"B"
```

Index

Instructional Computing Fundamentals for the **Commodore 64®**

ISBN 0-534-04662-2